(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: **10228380 A**

(43) Date of publication of application: **25 . 08 . 98**

(51) Int. Cl **G06F 9/44**

(21) Application number: **09184277**

(22) Date of filing: **05 . 06 . 97**

(30) Priority: **05 . 06 . 96 US 96 658472**

(71) Applicant: **SUN MICROSYST INC**

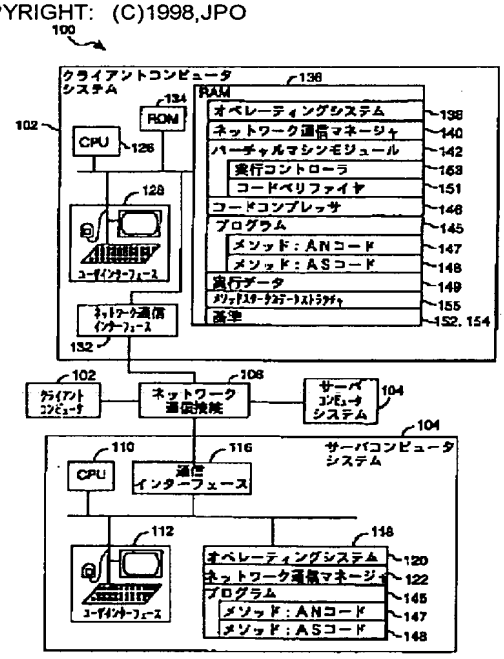
(72) Inventor: **LINDHOLM TIMOTHY G**

(54) METHOD AND COMPUTER SYSTEM FOR EXECUTING NETWORK MOVING CODE HAVING REDUCED RUN-TIME MEMORY SPACE REQUEST

(57) Abstract:

PROBLEM TO BE SOLVED: To reduce the necessary space capacity of a reduced run-time memory and to execute a network moving code.

SOLUTION: A client computer 102 can execute a program by reducing the necessary space capacity of the reduced run-time memory. A network communication interface 132 receives the method of the program, and at the time of receiving the method, a network communication manager 140 loads the method to the usable space of the run-time memory without compressing it. An execution controller 153 controls the execution of the program so that the method is accessed or not accessed at various time. A compressor 146 compresses a compressible method out of unaccessed and uncompressed methods in the memory so that the space can be utilized in the memory. The compressor 146 decompresses a decompressible method in the usable space of the memory so that the method is accessed.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平10－228380

(43)公開日　平成10年(1998)８月25日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/44	５３０	Ｇ０６Ｆ　9/44	５３０Ｋ

審査請求　有　請求項の数17　ＯＬ　外国語出願　（全 42 頁）

(21)出願番号　特願平9－184277

(22)出願日　平成９年(1997)６月５日

(31)優先権主張番号　０８／６５８４７２

(32)優先日　1996年６月５日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　594170738
サン　マイクロシステムズ　インコーポレイテッド
アメリカ合衆国　カリフォルニア州　94043　マウンテン　ヴィュー　ガルシア　アヴェニュー　2550

(72)発明者　ティモシー　ジー　リンドホーム
アメリカ合衆国　カリフォルニア州　94301　パロ　アルト　ミドルフィールド　ロード　623

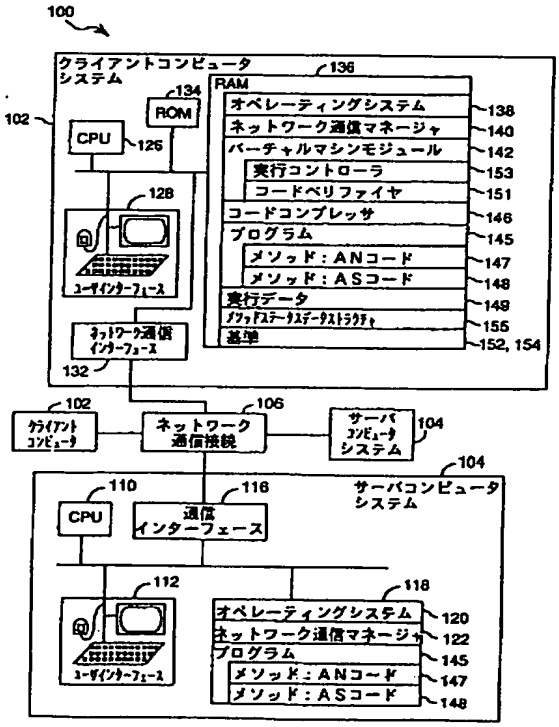
(74)代理人　弁理士　中村　稔　（外６名）

(54)【発明の名称】　低減されたラン－タイム・メモリ空間要求を有するネットワーク移動コードを実行する方法及びコンピュータ・システム

(57)【要約】　（修正有）

【課題】　低減されたランタイムメモリのスペース必要量を低くし、ネットワーク移動コードを実行する。

【解決手段】　クライアントコンピュータ１０２は低減されたランタイムメモリのスペース必要量を低くし、プログラムを実行できる。ネットワーク通信インターフェース１３２はプログラムのメソッドを受け取り、ネットワーク通信マネージャ１４０は、メソッドが受け取られたとき、メソッドをランタイムメモリの利用可能な空間に圧縮しないでロードする。実行コントローラ１５３は、プログラムの実行中、メソッドが種々の時間で呼び出されたり呼び出されないようにプログラムの実行を制御する。コンプレッサ１４６は、空間がランタイムメモリで利用可能なように、呼び出されていない圧縮されていないメソッドのうちの圧縮可能なものをメモリに圧縮する。コンプレッサは、メソッドが呼び出されうるように、メソッドのうちの解凍可能なものをランタイムメモリの利用可能な空間に解凍する。



【特許請求の範囲】

【請求項１】　メソッドを有するプログラムが設けられているコンピュータネットワークにおいて、ランタイムメモリのスペース必要量を低くして、該プログラムを実行するクライアントコンピュータシステムであって、
ランタイムメモリと、
メソッドを受け取るネットワーク通信インターフェースと、
受け取った場合に、ランタイムメモリの利用可能なスペースにメソッドを圧縮しないでロードするネットワーク通信マネージャと、
前記プログラムの実行を制御して、異なる時間において前記メソッドが呼び出されたり呼び出されないようにする、実行コントローラと、
（Ａ）呼び出されていない非圧縮メソッドのうちの圧縮可能なものをランタイムメモリにおいて圧縮し、それにより前記ランタイムメモリにおけるスペースを利用可能とし、（Ｂ）前記圧縮されたメソッドの解凍可能なものが呼び出されうるように、前記圧縮されたメソッドの解凍可能なものをランタイムメモリの利用可能なスペースにおいて解凍する、コンプレッサと、を有するクライアントコンピュータシステム。

【請求項２】　前記コンプレッサは、前記圧縮されたメソッドの解凍可能なものが呼び出されるとすぐに、前記圧縮されたメソッドの解凍可能なものを解凍する、請求項１に記載のクライアントコンピュータシステム。

【請求項３】　前記コンプレッサは、所定の時間間隔後に圧縮されたメソッドの解凍可能なものを解凍する、請求項１に記載のコンピュータシステム。

【請求項４】　前記コンプレッサは、圧縮されていないメソッドの圧縮可能なものがもはや呼び出されない状態になるとすぐに、圧縮されていないメソッドの圧縮可能なものを圧縮する、請求項１、２又は３に記載のクライアントコンピュータシステム。

【請求項５】　前記コンプレッサは、前記ランタイムメモリ内のスペースが必要であるが利用可能でない場合、圧縮されていないメソッドの圧縮可能なものを圧縮する、請求項１、２又は３に記載のクライアントコンピュータシステム。

【請求項６】　最も古く呼び出されたメソッドから最も新しく呼び出されたメソッドまでの順番に、現在呼び出されていない前記メソッドのそれらをリストする最も古く呼び出されたリスト、を更に備え、
前記圧縮されていないメソッドの圧縮可能なものは、前記ランタイムメモリのスペースが必要であるが利用可能でない場合、最も古く実行されたリストにおける最も古く実行されたメソッドのうちの圧縮されていないものである、請求項１乃至５のいずれか１項に記載のクライアントコンピュータシステム。

【請求項７】　前記メソッドは、クライアントコンピュータシステムのアーキテクチャに依存しないアーキテクチャニュートラルコードにおけるメソッドを含み、
前記クライアントコンピュータシステムが、実行コントローラを含み且つアーキテクチャニュートラルコードにおけるメソッドの実行を可能にするバーチャルマシンモジュールを更に備える、請求項１乃至６のいずれか１項に記載のクライアントコンピュータシステム。

【請求項８】　前記コンプレッサは、前記ランタイムメモリのスペースが必要であるが利用可能でない場合、圧縮されたメソッドのうちフラッシュ可能なものをランタイムメモリからフラッシュする、請求項１乃至７のいずれか１項に記載のクライアントコンピュータシステム。

【請求項９】　２次メモリと、
（Ａ）前記ランタイムメモリのスペースが必要であるが利用可能でない場合、前記圧縮されたメソッドのうち格納可能なものを前記２次メモリに格納し、且つ、（Ｂ）解凍されるべき前記圧縮されたメソッドのうち検索可能なものを２次メモリから検索する、コンプレッサと、を更に有する、請求項１乃至８のいずれか１項に記載のクライアントコンピュータシステム。

【請求項１０】　メソッドを有するプログラムが設けられているコンピュータネットワークにおいて、ランタイムメモリのスペース必要量を少なくして、該プログラムを実行する方法であって、
ランタイムメモリを設け、
メソッドを受け取り、
前記メソッドが受け取られたとき、該メソッドをランタイムメモリの利用可能なスペースに圧縮しないでロードし、
前記プログラムを実行して、前記メソッドが異なる時間で呼び出されたり呼び出されなかったりするようにし、
呼び出されていない非圧縮メソッドのうち圧縮可能なものをメモリ内において圧縮し、それにより前記ランタイムメモリにおいてスペースを利用可能にし、
圧縮されたメソッドのうちの解凍可能なものが呼び出されうるように、前記圧縮されたメソッドのうちの解凍可能なものをランタイムメモリの利用可能なスペースに解凍する、段階を有することを特徴とする方法。

【請求項１１】　前記解凍段階は、圧縮されたメソッドのうち解凍可能なものが呼び出されるべきときになるとすぐに、圧縮されたメソッドのうち解凍可能なものを解凍することを含む請求項１０に記載の方法。

【請求項１２】　前記解凍段階は、所定の時間間隔後に圧縮されたメソッドのうちの解凍可能なものを解凍することを含む、請求項１０に記載の方法。

【請求項１３】　前記圧縮段階は、前記圧縮されたメソッドのうち圧縮可能なものがもはや呼び出されない状態になるとすぐに、圧縮されていないメソッドのうちの圧縮可能なものを圧縮することを含む、請求項９，１１又は１２に記載の方法。

【請求項１４】 前記圧縮段階は、前記ランタイムメモリのスペースが必要であるが利用可能でない場合に、前記圧縮されていないメソッドのうちの圧縮可能なものを圧縮することを含む、請求項１０，１１又は１２に記載の方法。

【請求項１５】 最も古く呼び出されたメソッドから最も新しく呼び出されたメソッドまでの順番に現在呼び出されていない前記メソッドのそれらをリストする最も古く呼び出されたリストを設け、

前記圧縮されていないメソッドのうちの圧縮可能なものは、前記ランタイムメモリのスペースが必要であるが利用可能でない場合に、圧縮されていない最も古く呼び出されたリストにおける前記最も古く呼び出されたものとする、請求項１０乃至１４のいずれか１項に記載の方法。

【請求項１６】 前記ランタイムメモリのスペースが必要であるが利用可能でない場合に、圧縮されたメソッドのうちのフラッシュ可能なものを前記ランタイムメモリからフラッシングする段階を更に含む、請求項１０乃至１５のいずれか１項に記載の方法。

【請求項１７】 ２次メモリを設け、

前記ランタイムメモリのスペースが必要であるが利用可能でない場合に、圧縮されたメソッドのうちの格納可能なものを前記２次メモリに格納し、

解凍されるべき前記圧縮されたメソッドのうちの検索可能なものを２次メモリから検索する、段階を更に含む請求項１０乃至１６のいずれか１項に記載の方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、ランタイムメモリのスペース必要量を低くして、該プログラムを実行するコンピュータシステム及び方法に関する。特に、本発明は、アーキテクチャ特定コードのランータイムメモリのスペース必要量が低減されるようにネットワーク上に伝送されたコードから生成されたコードを実行するネットワークにおけるコンピュータ・システム及び方法に関する。

【０００２】

【従来の技術】今、コンピュータシステムは、コードが、以後ＡＮコードと称する、アーキテクチャニュートラル（ＡＮ）２進フォーマットとなったプログラムの特性を利用すべく構築または構成されている。かくして、これらのプログラムのＡＮコードは、コンピュータシステムの特定アーキテクチャまたはプラットフォームから独立である。本明細書の目的に関しては、アーキテクチャという用語は、一系列のコンピュータモデルのオペレーティング特性を意味するものとして定義される。特定のアーキテクチャの例としては、マッキントッシュコンピュータ、ＤＯＳまたはＷｉｎｄｏｗｓオペレーティングシステムを用いているＩＢＭ　ＰＣコンパチブルコンピュータ、Ｓｏｌａｒｉｓオペレーティングシステムを走らせているサン・マイクロシステムズ・コンピュータ、及びＵｎｉｘオペレーティングシステムを用いているコンピュータシステムがある。

【０００３】アーキテクチャスペシフィック（ＡＳ）という用語は、本明細書の目的に関しては、特定のコンピュータアーキテクチャを有するコンピュータシステムでだけの実行のために、あるプログラムのコードが、以後ＡＳコードと称する、２進フォーマットの形態であるという要求を指すものとして、定義される。かくして、通常のプログラミング言語（例えば、Ｃ言語）で記述され且つ特定のアーキテクチャ（例えば、ＩＢＭコンパチブルＰＣ）に対してコンパイルされたコードを有するプログラムは、そのアーキテクチャまたはそのアーキテクチャのエミュレータでだけ実行することができる。本明細書の目的において、アーキテクチャニュートラル（ＡＮ）という用語は、コンパイルされたコードを異なるアーキテクチャを有する種々のコンピュータシステムで実行することができるプログラムを指すものとして、定義される。例えば、特定のアーキテクチャを有するコンピュータシステムは、Ｊａｖａ（サン・マイクロシステムズの登録商標）バーチャルマシンモジュールで構成することができる。Ｊａｖａバーチャルマシンモジュールは、Ｊａｖａバーチャルマシンの命令セットに対して、以後、Ｊａｖａバイトコードと称する、Ｊａｖａプログラミング言語で記述され且つバイトコードにコンパイルされたコードを有するプログラムの実行を可能にする。Ｊａｖａバイトコードは、コンピュータシステムの特定のアーキテクチャから独立である。

【０００４】ＡＮコードを有するプログラムの重要な特徴には、それらの移植性がある。例えば、ＡＮコードのプログラムは、コンピュータシステムの特定アーキテクチャに関係なくＡＮコードを実行するために構成されたいかなるコンピュータシステムでも実行することができるので、これらのプログラムは、一つのコンピュータシステムから別のコンピュータシステムにネットワーク上を容易に移送することができる。例えば、Ｊａｖａバイトコードにコンパイルされたプログラムは、Ｊａｖａバーチャルマシンモジュールを有するいかなるコンピュータシステムでも実行することができ、かつＨｏｔＪａｖａ（サン・マイクロシステムズの登録商標）ネットワーク通信マネージャを用いて一つのコンピュータシステムから別のコンピュータシステムにネットワーク上を容易に移送することができる。

【０００５】更に、Ｊａｖａバイトコードにコンパイルされたプログラムの移植性に関する別の重要な特徴は、かかるプログラムの検証可能性である。特に、Ｊａｖａバーチャルマシンモジュールは、これらのプログラムが所定の完全性基準を満足するということを容易に検証することができる。かかる完全性基準は、Ｊａｖａバイトコードがバーチャルマシンモジュールのスタッ

クをオーバフローまたはアンダフローすることができないということ、及び、Ｊａｖａバイトコードの全ての命令は、そのデータタイプがそれらの命令に対するデータタイプ制限と合うデータだけを利用するということを保証するスタック及びデータタイプ使用制限を含む。結果として、Ｊａｖａバイトコードのプログラムは、オブジェクトポインタをフォージ(forge) することができず、且つ、一般的には、ユーザが使用することの許可を明示的に認可したもの以外のシステムリソースをアクセスすることができない。

【０００６】これらの理由のため、コンピュータシステムは、ネットワーク上で受け取られるＡＮコードのプログラムの実行のために構成されている。実際に、ある場合では、プログラムがコンピュータシステムのランタイム（即ち、実行時間）メモリ（例えば、ランダムアクセスメモリ（ＲＡＭ））に直接ロードされるので、かかるコンピュータシステムは、二次メモリ（例えば、ハードディスク）さえも必要としない。結果として、かかるコンピュータシステムのユーザは、現在ソフトウェアプロダクトの典型であるソフトウェア購入、設置、構成及びアップグレードのサイクルから解放される。上記において説明したＡＮコードの特徴は、ネットワーク化され且つ要求に応じてＡＮコードでロードされる小さいまたは安いコンピュータシステムでの使用を特に魅力的にする。例えば、これらの種類のコンピュータシステムは、ビデオゲーム、パーソナルディジタルアシスタント（ＰＤＡｓ）、セルラフォン、若しくは、他の同様なコンピュータシステム又はコンピュータ作動の装置であってよい。

【０００７】しかしながら、あいにく、ＡＮコードのプログラムは、ＡＳコードの同じプログラムよりも作動が遅い。例えば、Ｊａｖａバーチャルマシンモジュールによって実行されるＪａｖａバイトコードのプログラムは、典型的には、ＡＳコードの同等プログラムの２．５から２０倍ほど低速である。従って、コンピュータシステムのユーザは、コンピュータシステムの特定アーキテクチャでＡＳコードを実行できるようにプログラムのＡＳコードをネットワーク上で受け取るのが望ましいことを見出すようになる。しかし、これらのプログラムがコンピュータシステムに安全に送られることを保証するためには、ネットワークは閉鎖的であるか、信用保証されていなければならず、若しくは、これらのプログラムは検証することができる内蔵ディジタル署名を有しなければならない。

【０００８】更に、ＡＮから生成されたＡＳコードは最初のＡＮコードよりもかなり大きい。例えば、Ｊａｖａバイトコードから生成されたＡＳコードは、典型的には、Ｊａｖａバイトコードの大きさの２乃至５倍である。従って、一定容量のランタイムメモリは、ＡＮコードよりも大幅に少ないコンパイルされたＡＳコードを保持することができるにすぎない。しかしながら、前述したように、ＡＳコードは、ＡＳコードを生成するＡＮコードよりかなり速く、十分な性能を達成する唯一の方法でありうる。上述したコンピュータシステムでは、価格が非常に重要である。実際には、かかるコンピュータシステムを構築するにあたって最も重要なコストの一つは、ロードされたプログラムの実行のために要求されるランタイムメモリの量である。従って、そのような低減が強力な競合し得る利点を生み出すので、これらのコンピュータシステムによって要求されるランタイムメモリの量を低減することは、非常に重要である。

【０００９】更に、上述の形式のコンピュータシステムでは、二次メモリにページすることが可能でないかもしれない。この場合では、ネットワーク上で受け取られたＡＮコード又はＡＳコードは、ランタイムメモリにキャッチされ、ランタイムメモリにおけるスペースが必要とされる場合にフラッシュされることになる。しかしながら、フラッシュされたプログラムの実行が継続されるべき場合、オリジナルコードは、ネットワーク上に再びダウンロードされなければならない。このことは、プログラムの実行スピードにかなり影響を及ぼす。更に、二次メモリにページすることが可能なコンピュータシステムにおいてさえも、二次メモリからコードを検索するために必要な時間は、費用がかかりすぎるであろう。本発明では、ネットワーク上で受け取られたプログラムのコードの圧縮そして解凍が、ランタイムメモリにおけるコードの記憶費用を低減するのに用いられる。これは、コードをフラッシュしかつそれを検索することよりもかなり速いので、コードの実行スピードは、圧縮及び解凍によって著しく影響されない。

【００１０】

【課題を解決するための手段】まとめると、本発明は、メソッドを有するプログラムがコンピュータネットワークに与えられるようになったクライアントコンピュータシステムと、それに関連するコンピュータネットワークにおける関連方法である。クライアントコンピュータは、低いランタイムメモリのスペース必要量でプログラムを実行することができる。クライアントコンピュータシステムは、ランタイムメモリ、通信インターフェイス、ネットワーク通信マネージャ、実行コントローラ、及びコンプレッサを備えている。ネットワーク通信インターフェイスはプログラムのメソッドを受信し、ネットワーク通信マネージャは受け取った場合に圧縮されていないメソッドをランタイムメモリの利用可能なスペースにロードする。実行コントローラは、メソッドが呼び出され及び呼び出されないようにプログラムの実行を制御する。

【００１１】コンプレッサは、呼び出されない圧縮されたプログラムの圧縮可能なものをメモリにおいて圧縮する。その結果、ランタイムメモリのスペースが利用可能

になる。コンプレッサはまた、メソッドの解凍可能なものが呼び出されうるように、圧縮されたメソッドの解凍可能なものをランタイムメモリの利用可能なスペースに解凍する。一実施形態では、コンプレッサは、メソッドの解凍可能なものが呼び出されるとすぐに圧縮されたメソッドの解凍可能なものを解凍する。別の実施形態では、コンプレッサは、所定の時間間隔の後で圧縮されたメソッドの解凍可能なものを解凍する。更に別の実施形態では、コンプレッサは、メソッドの圧縮可能なものがもはや呼び出されない状態になるとすぐに、圧縮されていないメソッドの圧縮可能なものを圧縮する。

【００１２】更に別の実施形態では、コンプレッサは、ランタイムメモリにおけるスペースが必要であるが利用可能でない場合に、圧縮されていないメソッドの圧縮可能なものを圧縮する。更に、この実施形態では、クライアントコンピュータシステムは、最も古く呼出されたメソッドから最も新しく呼出されたメソッドの順番に現在呼び出されていないメソッドのそれらをリストする最も古く呼出されたリストを更に備えうる。結果として、圧縮されていないメソッドの圧縮可能なものは、ランタイムメモリにおけるスペースが必要であるが利用可能でない場合に、最も古く呼出されたリストにおける最も古く呼出されたメソッドである。更に別の実施形態では、コンプレッサは、ランタイムメモリにおけるスペースが必要であるが利用可能でない場合に、圧縮されたメソッドのフラッシュ可能なものをランタイムメモリからフラッシュする。

【００１３】先の実施形態に対する変形実施形態として、クライアントコンピュータシステムは、二次メモリを更に備えうる。この場合では、コンプレッサは、ランタイムメモリにおけるスペースが必要であるが利用可能でない場合に、圧縮されたメソッドの記憶可能なものを二次メモリに格納する。次いで、コンプレッサは、解凍されるべきである圧縮されたメソッドの検索可能なものを二次メモリから検索する。本発明の更なる目的及び特徴は、図面を参照することにより、以下の詳細な説明及び特許請求の範囲からさらに容易に明らかになるであろう。

【００１４】

【発明の実施の形態】図１を参照すると、本発明によるコンピュータネットワーク100 が示されている。該コンピュータネットワーク100 は１又はそれ以上のクライアントコンピュータシステム102 と、１又はそれ以上のサーバコンピュータシステム104 と、ネットワーク通信接続手段106 と、を含む。クライアントコンピュータシステム102 は、ネットワーク通信接続106 を介してサーバコンピュータシステム104 に接続されている。ネットワーク通信接続は、ローカルエリアネットワーク又はワイドエリアネットワーク、インターネット、若しくは、他のタイプのネットワーク通信接続であってよい。各サーバコンピュータシステム104 は、中央演算処理装置（ＣＰＵ）110 と、ネットワーク通信インターフェース116 と、メモリ118 とを含む。ネットワーク通信インターフェースにより、各サーバコンピュータシステムは、ネットワーク通信接続106 を介してクライアントコンピュータシステム102 と接続することできる。

【００１５】各サーバコンピュータシステム104 のメモリ118 は、オペレーティングシステム120 と、ネットワーク通信マネージャ（即ちサーバ）122 と、プログラム145とを格納する。オペレーティングシステムと通信マネージャは全てＣＰＵ120 の上で実行される。オペレーティングシステムは、ユーザインターフェース112 でユーザによって発行されたコマンド、又は、クライアントコンピュータシステム102 のユーザからネットワーク通信接続106 を介してネットワーク通信インターフェース116 によって受けとられたコマンドに応じてネットワーク通信マネージャの走行を制御及び調整する。プログラム145 はメソッド147 及び／又は148 を備える。本明細書の目的のために、プログラムの実行中、種々の時間で呼び出され且つ呼び出されないプログラムのいかなる区分フラグメント即ち部分はメソッドと考えられる。

【００１６】各サーバコンピュータシステム104 のメソッド147 は、クライアントコンピュータシステム102 の特定のアーキテクチャ（即ちプラットフォーム）に依存しないアーキテクチャニュートラル（ＡＮ）コードを含む。これらのプログラムは特定のプログラミング言語からＡＮコードにコンパイルされる。好ましい実施形態では、これらのプログラムはＪａｖａプログラミング言語で記述され、Ｊａｖａバイトコードにコンパイルされる。更に、これらのプログラムは、オブジェクト指向手段でプログラムされたソフトウェアプログラムを形成するメソッドでオブジェクトクラスに含まれる。他方では、各サーバコンピュータシステムのメソッド148 は、クライアントコンピュータシステム102 の特定のアーキテクチャのためにコンパイルされたアーキテクチャスペシフィック（ＡＳ）コードを含む。先に言及したように、これらの種類のメソッドは、ＡＮコードの同じメソッドより速く実行できるので望ましい。後でより詳細に説明するように、ネットワーク100 は、これらのプログラムが高信頼度でクライアントコンピュータ102 に安全に出荷される閉じた、信頼性の高いネットワークであることが望ましく、また、これらのプログラムは確認することができる内蔵ディジタル署名を有するのが望ましい。

【００１７】後でまた、より詳細に説明するように、メソッド147 及び／又は148 は、ユーザがリクエストしたときに、ネットワーク通信マネージャ122 を使用するクライアントコンピュータシステム102 に伝送される。かくして、これらのメソッドのコードはネットワーク移動コードと考えられる。各クライアントコンピュータシス

テム102 は、ビデオゲーム、パーソナルディジタルアシスタント（ＰＤＡ）、セルラフォン、デスクトップコンピュータ、若しくは、他のコンピュータシステム、又は、少量のランータイムメモリを要求するコンピュータ作動デバイスであってよい。更に、各クライアントコンピュータシステムは中央演算処理装置（ＣＰＵ）126 、ユーザインターフェース128 、ネットワーク通信インターフェース132 、読み出し専用メモリ（ＲＯＭ）134 、ランータイムランダムアクセスメモリ（ＲＡＭ）136 とを含む。ネットワーク通信インターフェースにより、クライアントコンピュータシステムはネットワーク通信接続106 を介してサーバコンピュータシステム104 と通信することができる。

【００１８】各クライアントコンピュータシステム102 のＲＡＭ136 は、オペレーティングシステム138 、ネットワーク通信マネージャ140 、バーチャルマシンモジュール142 、及び、ＲＯＭ134 から全てロードされたコード圧縮プログラム146 を格納する。ＲＡＭはまた、ＡＮコードを含むメソッド147 、及び／又は、サーバコンピュータシステム104 からダウンロードされたＡＳコードを含むメソッド148 を有するプログラム145 を格納する。オペレーティングシステム、ネットワーク通信マネージャ、バーチャルマシンモジュール、コンパイラ、圧縮プログラム、及び、プログラムは全てＣＰＵ126 で実行される。オペレーティングシステムは、ユーザインターフェース128 でユーザによって発行されたコマンドに応じて、ネットワーク通信マネージャ、バーチャルマシンモジュール、コードコンパイラ、コード圧縮プログラム、及び、プログラムの実行を制御及び調整する。

【００１９】後で言及するように、メソッド147 及び／又は148 はユーザがリクエストしたときに、サーバコンピュータシステム104 から受信される。好ましい実施形態ではＨｏｔＪａｖａネットワーク通信マネージャであるネットワーク通信マネージャ122 を使用するこれらのメソッドが行われている。次いで、ネットワーク通信マネージャはＲＡＭ136 にこれらのメソッドをロードする。バーチャルマシンモジュール142 のコードベリファイヤ151 は、ロードされたメソッド147 のＡＮコードが所定の完全性基準にあうことを検証する。先に述べたように、このことは、ロードされたメソッドがバーチャルマシンモジュールのスタックをオーバーフロー又はアンダーフローすることができないように、及び、すべての命令がそれらの命令のためのデータタイプ制限に一致するデータタイプのデータのみを利用するように、スタック及びデータタイプ使用制限を含む。好ましい実施形態では、バーチャルマシンモジュールはＪａｖａバーチャルマシンモジュールである。

【００２０】しかしながら、ＡＳコードを有する受信されたメソッド148 の場合では、コードベリファイヤ151 はＡＳコードの完全性を直接検証するように使用することができない。かくして、ＡＳコードの完全性を間接的に検証するために、ネットワーク100 は、これらのメソッドが高信頼度でクライアントコンピュータ102 に安全に送出することができる、閉じた、信用ネットワークであるのがよい。変形実施形態として、ネットワーク100 が安全でなければ、これらのプログラムは、ネットワーク通信マネージャ140 が信頼できるソースからのものであることを検証することができる内蔵ディジタル署名を有するのがよい。バーチャルマシンモジュール142 の実行コントローラ153 はプログラム145 の実行を制御する。特に、実行コントローラは、クライアントコンピュータシステム102 の特定アーキテクチャで実行するためのメソッド147 のＡＮコードを翻訳し、実行コントローラによりこれらのメソッドは特定アーキテクチャで実行するためのＡＳコードを含むメソッド148 を呼び出すことができる。プログラムの実行中に生成された実行149 データはＲＡＭ136 に格納される。加えて、ネットワーク通信マネージャ140 、コードコンパイラ144 、及び／又は、コード圧縮プログラム146 がＡＮコードにあるならば、次いで、実行コントローラは同様にそれらの実行を制御する。

【００２１】更に、クライアントコンピュータシステム102 のＲＡＭスペース要求を低く保つために、コード圧縮プログラム146 は、メソッド147 及び／又は148 のコードを種々の時間でＲＡＭ136 に圧縮及び解凍する。該メソッドが、ＲＡＭ136 に格納され、且つ、ユーザインターフェース128 でユーザによって入力された、あらかじめ定められた圧縮及び解凍基準152 及び154 をそれぞれ満足させるので、このことは、圧縮でき且つ解凍できるメソッドのために行われる。圧縮及び解凍基準は、後でより十分に説明され、本発明の好ましい実施形態では、一連のあらかじめ定められたメモリ管理ストラテジから使用者が（ユーザインターフェース128 を使用して）選択可能及び／又は調整可能である。メソッド147 及び／又は148 のストレージ及び呼び出しステータスは、メソッドステータスデータストラクチャ155 で維持される。メソッドステータスデータストラクチャは、ネットワーク通信マネージャ140 、実行コントローラ153 、及び、コード圧縮プログラム146 によって更新される。

【００２２】図２は、メソッド147 及び／又は148 をＲＡＭ136 に圧縮又は解凍する際に、各クライアントコンピュータシステム102 の作動の機能ブロック図を示す。加えて、図３及び図４は好ましい圧縮メソッド及び解凍メソッド300 及び400 をそれぞれ示す。図１乃至図３を参照すると、ユーザがサーバコンピュータシステム104 の１つのプログラム145 の１つの実行を要求するとき、ユーザのクライアントコンピュータシステム102 は、要求されたプログラムをサーバコンピュータシステムから得る（図３のステップ302 ）。ユーザがプログラムをサ

ーバコンピュータシステムからダウンロード及び実行するために、ユーザインターフェース128 でコマンドを発行するとき、このことは行われる。それに応じて、オペレーティングシステム120 は、作られた要求を指示するメッセージを生成するネットワーク通信マネージャ140 を呼ぶ。次いで、ネットワーク通信インターフェース132 はメッセージをサーバコンピュータシステムに伝送する。

【００２３】サーバコンピュータシステム104 のネットワーク通信インターフェース116 は、伝送されたメッセージを受け取る。それに応じて、サーバコンピュータシステムのネットワーク通信マネージャ122 は要求されたプログラム145 のメソッド147 及／又は148 をネットワーク通信インターフェースに提供し、次いで、メソッドをユーザのクライアントコンピュータシステム102 に伝送する。伝送されたメソッド147 及び／又は148 は、ユーザのクライアントコンピュータシステム102 のネットワーク通信インターフェース132 によって受けとられる。それに応じて、ネットワーク通信マネージャ140 は、次いで、ＲＡＭ136 に受信されたメソッドをロードするのに十分なスペースがあるかどうかを判断する（図３の意志決定ステップ304 ）。もしＲＡＭスペースが利用できるならば、次いで、ネットワーク通信マネージャがＲＡＭの利用できるスペースに圧縮されていないこれらのメソッドのコードをロードする（図３のステップ306 ）。結果として、これらのメソッドは、以前にロードされた他のプログラム145 のプログラムメソッド147 及び／又は148 と一緒にＲＡＭにロードされる。これらのメソッドをロードする際に、ネットワーク通信マネージャは、メソッドと、これらのメソッドによって占有されたＲＡＭ136 のメモリスペースに対応するポインタとを確認するために、メソッドステータスデータストラクチャ155 のメソッドストレージステータステーブル200 を更新する。更に、ネットワーク通信マネージャは、メソッドのコードが圧縮されていない（Ｕ）ことを示すために、プログラムステータステーブルを更新する。

【００２４】次いで、ネットワーク通信マネージャ140 はバーチャルマシンモジュール142コードベリファイヤ151 を呼び出す。それに応じて、コードベリファイヤは、丁度ロードされたどんなメソッド147 のＡＮコードが以前に議論された事前定義保全基準にあうことを確認する（図３のステップ307 ）。次いで、メソッド147 及び／又は148 が丁度ロードされたプログラム145 は、以前にロードされたいかなるプログラムと一緒に、実行コントローラの制御下で実行される（図３のステップ314 ）。プログラムが実行されたとき、プログラムのメソッドは、それらの実行中、種々の時間で、呼び出され、且つ、呼び出されない。以前に述べたように、実行コントローラはユーザのクライアントコンピュータシステム102 の特定のアーキテクチャで実行するためのメソッド147 のＡＮコードを翻訳し、且つ、実行コントローラによってこれらのメソッドは特定のアーキテクチャで実行するためのＡＳコードを含むメソッド148 を呼ぶことができる。

【００２５】ロードされたメソッド147 及び／又は148 の各々は、それがデータのために待たなければならず、それがスリープさせられる、等のため、種々の時間で呼び出されないであろう。実行コントローラがこれが発生したということを決定する場合、それは、現在呼び出されていないということを示すためにメソッドをプログラムステータスデータ構造155 の最も古く呼び出された（ＬＲＩ）リスト202 に加える。ＬＲＩリストのメソッドは、最も新たらしく呼出されたものから最も古く呼出されたものまでリストされる。先に示したように、ＲＡＭ136 のスペース要求を低減するために、所定の圧縮基準152 が満足される各々ロードされたメソッド147 及び／又は148 のコードは、コードコンプレッサ146 によって圧縮される。好ましい実施形態では、圧縮基準は、圧縮可能メソッド147 又は148 のコードが（１）ＲＡＭ136 のスペースが必要であるが利用可能でなく、かつ（２）メソッドが、そのコードがまだ圧縮されていないＬＲＩリスト202 のメソッドの最も古く呼び出されたメソッドであるときに圧縮されるということを定める。

【００２６】ネットワーク通信マネージャ140 が、ＲＡＭ136 のスペースがサーバコンピュータシステム104 によって受け取られた１又はそれ以上のメソッド147 及び／又は148 をロードするのに利用可能ではないと判断する場合（図３の意思決定ステップ304 ）、次いで、それはコードコンプレッサ146 を呼出す。それに応じて、コードコンプレッサは、十分なスペースが利用可能になるまでそのコードがまだ圧縮されていないＬＲＩリスト202 の最も古く呼出されたメソッドのコードを圧縮する（図３のステップ316 ）。コードコンプレッサはまた、メソッドの圧縮されたコードのメモリスペースへの対応ポインタを識別し且つメソッドのコードが圧縮されたということを示すためにメソッドストレージステータステーブル200を更新する（Ｃ）。次いで、ネットワーク通信マネージャは、先に説明したように、利用可能なスペースにスペースが利用可能にされたメソッドをロードする（図３のステップ306 ）。

【００２７】コードコンプレッサ146 は、当業者によく知られたあらゆる高速データ圧縮技術を使用しうる。更に、コードコンプレッサは、メソッド147 のＡＮコード及びメソッド148 のＡＳコードの最適圧縮に対して個別の圧縮技術を使用しうる。更に、ロードされたメソッド147 及び／又は148 が呼び出されると同時に、それらは実行データを生成する。実行コントローラ153 が決定するこれらのメソッドのいずれかが呼び出されることに対し（図3 の意思決定ステップ320 ）、且つ、ＲＡＭ136 のスペースが実行データを格納するのに利用可能である

ことに対して（図３のステップ324 ）、実行コントローラは実行データをＲＡＭに格納する（図３のステップ324 ）。しかしながら、実行コントローラ153 が呼び出されたものと判断するメソッド147 及／又は148 であって（図３の意思決定ステップ320 ）、ＲＡＭのスペースが利用可能ではないが実行データを格納する必要がある（意思決定ステップ322）メソッドの各々について、実行コントローラはコードコンプレッサ146 を呼び出す。ＲＡＭのスペースが必要であるような先の状況におけるように、コードコンプレッサは、十分なスペースが利用可能になるまで、そのコードがまだ圧縮されていないＬＲＩリスト202 にある最も古く呼び出されたメソッドのコードを圧縮する（図３のステップ326 ）。そして、先に説明したように、コードコンプレッサはメソッドの圧縮されたコードのメモリスペースへの対応ポインタを識別し、且つ、メソッドのコードが圧縮されたということを示すために、メソッドストレージステータステーブル200 を更新する（Ｃ）。次いで、実行コントローラは、先に記述したように、実行データをＲＡＭにおいて利用可能になったスペースに格納し、その実行データが格納されたメソッドを有するプログラムの実行が継続される（図３のステップ314 ）。

【００２８】更に、図１、図２及び図４を参照すると、ＲＡＭ136 のスペースが利用可能である場合、圧縮され、且つ、所定の解凍基準154 が満足されるロードされた各メソッド147 及び／又は148 は、コードコンプレッサ146 によって解凍される。好ましい実施形態では、解凍可能なメソッドのコードを解凍するための所定の解凍基準は、メソッドのコードが圧縮され、且つ、メソッドが再び呼び出されるときに解凍されるということを単に特定する。かくして、実行コントローラ153 が、ロードされたメソッド147 及び／又は148 の１つが呼び出されるということを決定するときはいつでも（図４の意思決定ステップ402 ）、それはこのメソッドのコードが圧縮されるかどうかを決定する（図４の意思決定ステップ404 ）。これは、メソッドソトレージステータステーブル200 から決定される。メソッドのコードが圧縮されないならば、次いで、メソッドは実行コントローラによって呼び出される（図４のステップ406 ）。メソッドがもはや呼び出されない場合には、そのコードは先に説明した仕方でコードコンプレッサ146 によって圧縮されうる。しかしながら、呼び出されるメソッド147 及び／又は148 のコードが圧縮されるならば、次いで、実行コントローラはメソッドのコードを解凍するためにコードコンプレッサ146 を呼び出す。コードコンプレッサは、コードを解凍するためにＲＡＭ136 において十分なスペースが利用可能であるかどうかを決定する（図４の意思決定ステップ408 ）。十分なスペースが利用可能ならば、コードコンプレッサは利用可能なスペースでコードを解凍する（図４のステップ410 ）。そのようにすることで、それは、圧縮されていないコードのメモリスペースへの対応ポインタを識別し、且つ、今、メソッドのコードが圧縮されていないということを示すために、メソッドストレージステータステーブル200 を更新する（Ｕ）。

【００２９】しかしながら、十分なスペースが利用可能でないならば、次いで、コードコンプレッサ146 は、スペースが利用可能になるまでそのコードがまだ圧縮されていないＬＲＩリスト202 の最も古く実行されたメソッドのコードを圧縮する（図４のステップ412 ）。このことは先に示した仕方で行われる。スペースが利用可能になった後、解凍されるべきメソッド147 又は148 のコードは、利用可能なスペースでコードコンプレッサによって解凍され、次いで、丁度説明した仕方で実行コントローラ153 よって呼び出される（図４のステップ410 及び406 ）。前記の観点では、本発明が、実行スピードをセーブすると同時にランタイムメモリスペースの低減を提供することは明らかである。これは、ロードされたメソッド147 又は148 を圧縮することによって、メソッド147 又は148 はＲＡＭにおけるスペースが必要な場合に、ＲＡＭ136 からフラッシュされ且つサーバコンピュータシステム104 から再ダウンロードされる必要がなくはないという事実による。しかしながら、当業者が認識するように、他の変形実施形態は同様な利益を提供すべくインプリメントすることができる。

【００３０】特に、先に説明した圧縮基準152 は、ＲＡＭ136 のスペースが必要であったが利用可能でない場合に圧縮されていないコードを有する最も古く呼び出されたメソッド147 又は148 のコードが圧縮されるであろうということを特定した。しかしながら、圧縮基準は、広範囲のオプションからユーザによって選択され、且つ、ユーザのクライアントコンピュータシステム102 に多数の条件に基づいてよい。例えば、圧縮基準は、各メソッドのコードが、メソッドがもはや呼び出されなくなったらすぐに圧縮されるということを単に特定しうる。若しくは、圧縮基準は、所定のメソッドのコードがそのようにするための時間が利用可能なときはいつでものらくらと圧縮されるということを特定しうる。先のいずれかのものの更なる変形として、圧縮基準は特定の大きさ又はタイプのメソッドのコードだけが圧縮されるということを特定しうる。

【００３１】更に、先に示したように、解凍基準154 は、そのコードが圧縮されたコードであるメソッド147 又は148 が、メソッドが呼び出されるとすぐにそのコードを解凍させるということを特定した。しかしながら、解凍基準は、所定の時間間隔が満了になった後に、圧縮されたこーどが解凍されることを特定できる。この場合では、データコンプレッサは時間間隔を計るためのタイマを含む。一例では、この技術は、メソッドが既知の時間間隔に対して圧縮され、次いで、メソッドが目覚めさせられるときのほんの少し前に解凍されるように、既知

の時間間隔に対してスリープさせられているメソッドに用いることができる。若しくは、別の例では、データがメソッドにとって利用可能になるときを予測してメソッド圧縮の時間間隔が選択されるような場合に、データを待っているメソッドに対してこの技術を用いることができる。

【００３２】別の実施形態では、圧縮基準152 はまた、ロードされたメソッド147 又は148がＲＡＭ136 からフラッシュされる場合を特定しているフラッシング基準も含んでよい。例えば、フラッシィング基準は、呼び出されていない且つ圧縮されていないＲＡＭにロードされたメソッドが存在しないならば、次いで、圧縮されたコードを有するＬＲＩリスト202 の最も古く呼び出されたメソッドは、フラッシュ可能であり、且つ、十分なスペースが利用可能になるまでＲＡＭ136 からフラッシュれるということを示しうる。結果として、メソッドがＲＡＭからフラッシュされ、次いで、再び実行される場合、ネットワーク通信マネージャ140 は先に議論した仕方でプログラムを供給するサーバコンピュータシステム104 からプログラムを再ダウンロードしなければならない。実行コントローラ153 は、フラッシュされたいかなるメソッドへの参照を除去することによってプログラムストレージステータステーブル200 を更新する。

【００３３】ＲＡＭ136 からロードされたメソッド147 又は148 をフラッシュすることは実行スピードに関して費用がかかるので、図５及び図６に示すように、２次メモリ500 は、さもなければフラッシュされるメソッドを格納するのに使用することができる。この場合では、先に議論した圧縮基準152 は、丁度議論したフラッシング基準に類似し、且つ、それらが２次ストレージ基準を満足するので格納可能であるメソッドを格納するのに使用される、２次ストレージ基準を含む。しかしながら、この場合では、コードコンプレッサ146 は、これらのメソッドを２次メモリにポイントするためにプログラムストレージステータステーブル200 のポインを更新する。更に、メソッドのコードが圧縮され、２次メモリに格納され、且つ、解凍されるべきであるように検索可能であるメソッドに関して、メソッドのコードは２次メモリから検索され、次いで、先に説明した仕方でＲＡＭ136 に解凍れる。

【００３４】更に、クライアントコンピュータシステム102 が２次メモリ500 を含むような施形態（例えば、ネットワークされたデスクトップコンピュータ）では、メソッド147 又は148 をサーバコンピュータシステムから２次メモリにダウンロードすることができる。次いで、これらのメソッドを、サーバコンピュータシステム104 からではなく、２次メモリからＲＡＭ136 に直接ロードすることができる。更に、かかる実施形態では、オペレーティングシステム138 、ネットワーク通信マージャ140 、バーチャルマシンモジュール142 、及び、コードコンプレッサ146を、２次メモリに格納し、そこからＲＡＭにロードすることができる。更に別の実施形態では、オペレーティングシステム138 、ネットワーク通信マージャ140 、バーチャルマシンモジュール142 、及び、コードコンプレッサ146を、サーバコンピュータシステム104 の１つからクライアントコンピュータシステム102 のＲＡＭ136 にダウンロードすることができる。これは、サーバコンピュータシステムのメソッド147 又は148 に対して先に説明されたものと同様な仕方で行われる。

【００３５】更に別の実施形態では、バーチャルマシンモジュール142 はシリコンチップで実際に実行され、クライアントコンピュータシステム102 のＣＰＵとしてサーブする。この場合では、ＡＮコードを有するメソッド147 は特定アーキテクチャのために翻訳されず、その代わりに、直接実行される。この実施形態では、ＡＳコードを有するメソッド148 は利用されない。最後に、本発明を２、３の特定の実施形態を参照して記述したが、記述は本発明の説明のためであり、本発明を限定するように解釈すべきでない。種々の変形が、特許請求の範囲によって定義されたような本発明の真の精神及び範囲から逸脱することなく当業者において行われてよい。

【図面の簡単な説明】

【図１】本発明を具体化したクライアントコンピュータシステムのブロック図である。

【図２】クライアントコンピュータシステムのオペレーションの機能ブロック図である。

【図３】クライアントコンピュータシステムの圧縮方法のフローチャートである。

【図４】クライアントコンピュータシステムの解凍方法のフローチャートである。

【図５】クライアントコンピュータシステムの変形実施形態を示す。

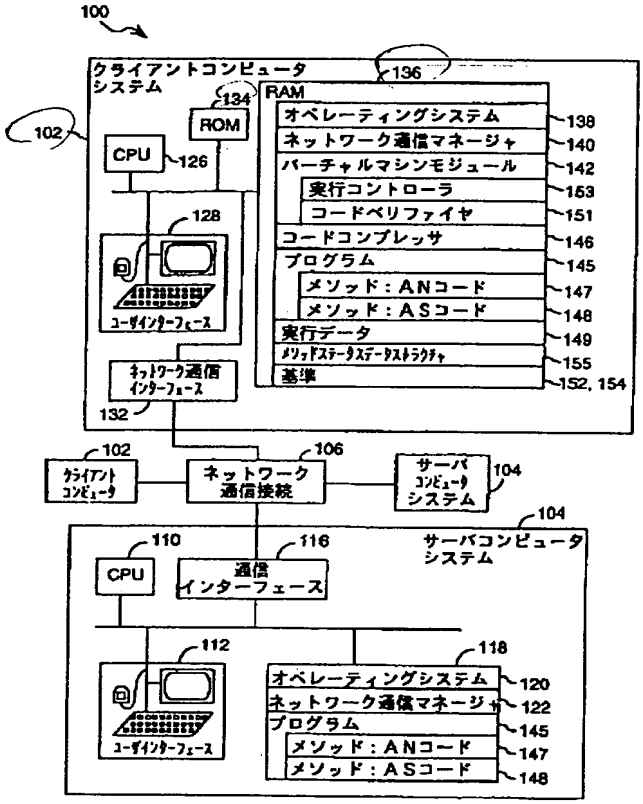
【図６】クライアントコンピュータシステムの変形実施形態のオペレーションの機能ブロック図を示す。

【符号の説明】

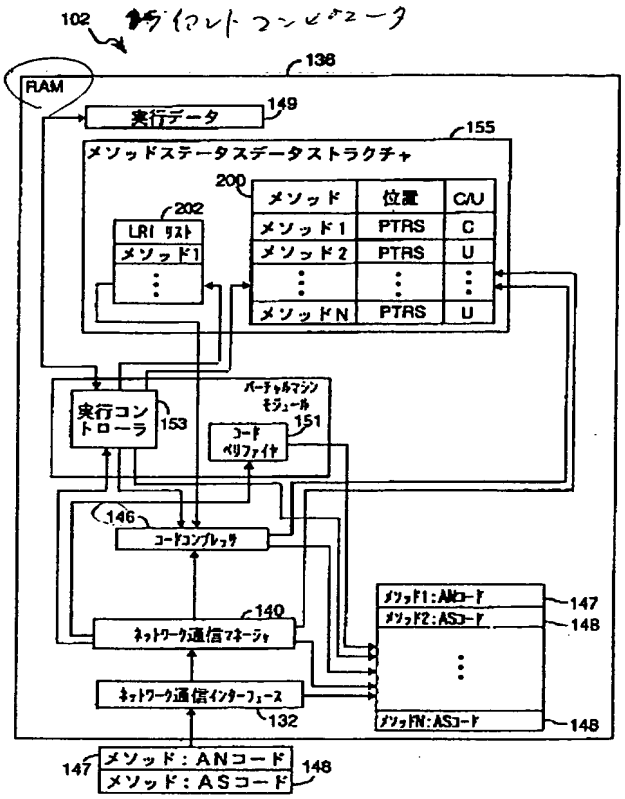
100　コンピュータネットワーク
102　クライアントコンピュータシステム
104　サーバコンピュータシステム
106　ネットワーク通信接続
110 、126　ＣＰＵ
112 、128　ユーザインターフェース
116　通信インターフェース
118　メモリ
128 、138　オペレーティングシステム
122 、140　ネットワーク通信マネージャ
132　ネットワーク通信インターフェース
134　ＲＯＭ
136　ＲＡＭ
142　バーチャルマシンモジュール

145　プログラム
146　コードコンプレッサ
147　メソッド：ＡＮコード
148　メソッド：ＡＳコード
149　実行データ
150、152、154　基準
151　コードベリファイヤ
153　実行コントローラ
155　メソッドステータスデータストラクチャ

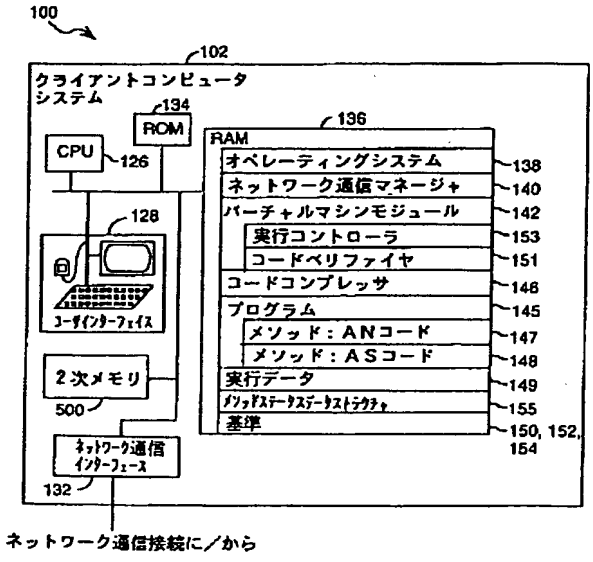
【図１】



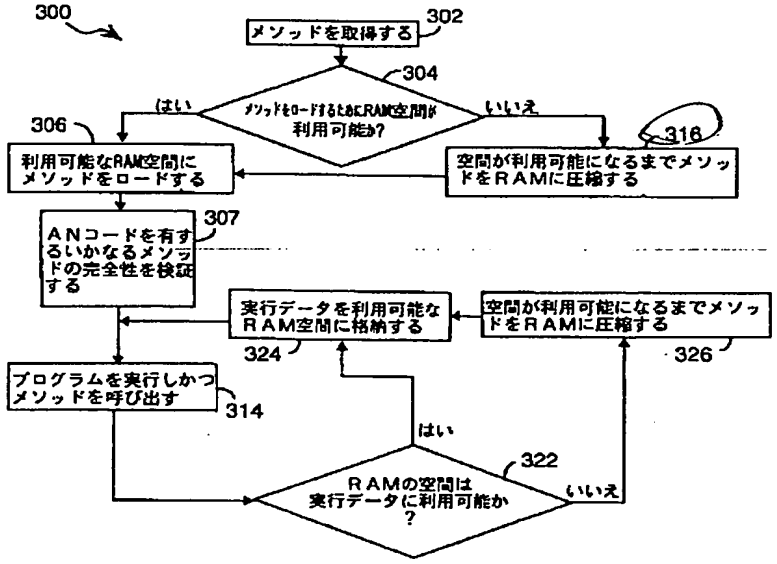
【図２】



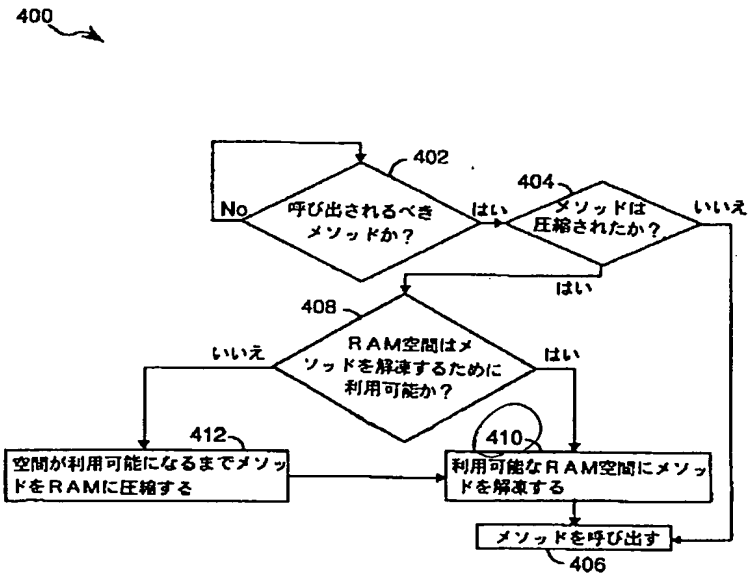
【図５】



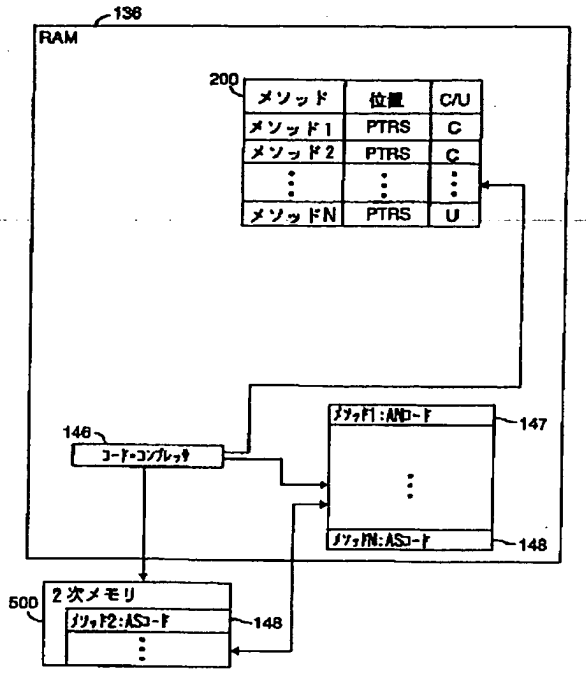
【図３】


300
メソッドを取得する
302
304
メソッドをロードするときにRAM空間が利用可能か?
はい
いいえ
306
利用可能なRAM空間にメソッドをロードする
316
空間が利用可能になるまでメソッドをRAMに圧縮する
307
ANコードを有するいかなるメソッドの完全性を検証する
実行データを利用可能なRAM空間に格納する
324
空間が利用可能になるまでメソッドをRAMに圧縮する
326
プログラムを実行しかつメソッドを呼び出す
314
はい
322
RAMの空間は実行データに利用可能か?
いいえ

【図４】


400
402
No
呼び出されるべきメソッドか?
はい
404
メソッドは圧縮されたか?
いいえ
はい
408
RAM空間はメソッドを解凍するために利用可能か?
いいえ
はい
412
空間が利用可能になるまでメソッドをRAMに圧縮する
410
利用可能なRAM空間にメソッドを解凍する
メソッドを呼び出す
406

【図６】



【手続補正書】
【提出日】平成１０年３月６日
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】特許請求の範囲
【補正方法】変更
【補正内容】
【特許請求の範囲】
【請求項１】　メソッドを有するプログラムが設けられているコンピュータネットワークにおいて、ランタイムメモリのスペース必要量を低くして、該プログラムを実行するクライアントコンピュータシステムであって、
ランタイムメモリと、
メソッドを受け取るネットワーク通信インターフェースと、
メソッドを受け取った場合に、ランタイムメモリの利用可能なスペースにメソッドをロードするネットワーク通信マネージャと、
前記プログラムの実行を制御して、異なる時間において前記各メソッドが呼び出されたり呼び出されないようにする、実行コントローラと、
（Ａ）選択されたメソッドが呼び出されておらず、且つ、圧縮されていないとき、メソッドの選択されたメソッドをランタイムメモリにおいて圧縮し、それにより前記ランタイムメモリにおけるスペースを利用可能とし、
（Ｂ）前記選択されたメソッドが呼び出されうるように、前記選択されたメソッドが圧縮されるとき、前記選択されたメソッドをランタイムメモリの利用可能なスペース内に解凍する、コンプレッサと、を有するクライアントコンピュータシステム。
【請求項２】　前記コンプレッサは、前記選択されたメソッドが呼び出されるとすぐに、前記選択されたメソッドを解凍する、請求項１に記載のクライアントコンピュータシステム。
【請求項３】　前記コンプレッサは、所定の時間間隔後に選択されたメソッドを解凍する、請求項１に記載のコンピュータシステム。
【請求項４】　前記コンプレッサは、選択されたメソッドがもはや呼び出されない状態になるとすぐに、選択されたメソッドを圧縮する、請求項１、２又は３に記載のクライアントコンピュータシステム。
【請求項５】　前記コンプレッサは、前記ランタイムメモリ内のスペースが必要であるが利用可能でない場合、選択されたメソッドを圧縮する、請求項１、２又は３に記載のクライアントコンピュータシステム。
【請求項６】　最も古く呼び出されたメソッドから最も新しく呼び出されたメソッドまでの順番に、現在呼び出されていない前記メソッドのそれらをリストする最も古く呼び出されたリスト、を更に備え、

前記選択されたメソッドは、前記ランタイムメモリのスペースが必要であるが利用可能でない場合、最も古く実行されたリストにおける最も古く実行されたメソッドのうちの圧縮されていないものである、請求項１乃至５のいずれか１項に記載のクライアントコンピュータシステム。

【請求項７】　前記メソッドは、クライアントコンピュータシステムのアーキテクチャに依存しないアーキテクチャニュートラルコードにおけるメソッドを含み、

前記クライアントコンピュータシステムが、（Ａ）実行コントローラを含み、（Ｂ）前記クライアントコンピュータシステムで走り、（Ｃ）アーキテクチャニュートラルコードにおけるメソッドの実行を可能にするバーチャルマシンモジュールを更に備える、請求項１乃至６のいずれか１項に記載のクライアントコンピュータシステム。

【請求項８】　前記コンプレッサは、選択されたメソッドが圧縮され、前記ランタイムメモリに更なるスペースが必要であるが利用可能でない場合、選択されたメソッドをランタイムメモリからフラッシュする、請求項１乃至７のいずれか１項に記載のクライアントコンピュータシステム。

【請求項９】　２次メモリと、

（Ａ）選択されたメソッドが圧縮され、前記ランタイムメモリに更なるスペースが必要であるが利用可能でない場合、前記選択されたメソッドを前記２次メモリに格納し、且つ、（Ｂ）選択されたメソッドが解凍されるべきである場合、前記選択されたメソッドを２次メモリから検索する、コンプレッサと、を更に有する、請求項１乃至８のいずれか１項に記載のクライアントコンピュータシステム。

【請求項１０】　メソッドを有するプログラムが設けられているコンピュータネットワークにおいて、ランタイムメモリのスペース必要量を少なくして、クライアントコンピュータシステムで該プログラムを実行する方法であって、

クライアントコンピュータでコンピュータネットワークからメソッドを受け取り、

前記メソッドが受け取られたとき、該メソッドをクライアントコンピュータシステムのランタイムメモリの利用可能なスペースにロードし、

前記プログラムを前記クライアントコンピュータシステムで実行して、前記メソッドが異なる時間で呼び出されたり呼び出されなかったりするようにし、

呼び出されておらず、且つ、圧縮されていないメソッドのうちの選択されたメソッドをランタイムメモリ内において圧縮し、それにより前記ランタイムメモリにおいてスペースを利用可能にし、

前記選択されたメソッドが呼び出されうるように、前記選択されたメソッドが圧縮されるとき、前記選択されたメソッドをランタイムメモリの利用可能なスペースに解凍する、段階を有することを特徴とする方法。

【請求項１１】　前記解凍段階は、選択されたメソッドが呼び出されるべきときになるとすぐに、選択されたメソッドを解凍することを含む請求項１０に記載の方法。

【請求項１２】　前記解凍段階は、所定の時間間隔後に選択されたメソッドを解凍することを含む、請求項１０に記載の方法。

【請求項１３】　前記圧縮段階は、前記選択されたメソッドがもはや呼び出されない状態になるとすぐに、選択されたメソッドを圧縮することを含む、請求項１０、１１又は１２に記載の方法。

【請求項１４】　前記圧縮段階は、前記ランタイムメモリのスペースが必要であるが利用可能でない場合に、前記選択されたメソッドを圧縮することを含む、請求項１０，１１又は１２に記載の方法。

【請求項１５】　最も古く呼び出されたメソッドから最も新しく呼び出されたメソッドまでの順番に現在呼び出されていない前記メソッドのそれらをリストする最も古く呼び出されたリストを設け、

前記選択されたメソッドは、前記ランタイムメモリのスペースが必要であるが利用可能でない場合に、圧縮されていない最も古く呼び出されたリストにおける前記最も古く呼び出されたものとする、請求項１０乃至１４のいずれか１項に記載の方法。

【請求項１６】　前記選択されたメソッドが圧縮され、前記ランタイムメモリに更なるスペースが必要であるが利用可能でない場合に、選択されたメソッドを前記ランタイムメモリからフラッシングする段階を更に含む、請求項１０乃至１５のいずれか１項に記載の方法。

【請求項１７】　前記選択されたメソッドが圧縮され、前記ランタイムメモリに更なるスペースが必要であるが利用可能でない場合に、選択されたメソッドをクライアントコンピュータシステムの前記２次メモリに格納し、前記選択されたメソッドが解凍されるべきである場合、前記選択されたメソッドを２次メモリから検索する、段階を更に含む請求項１０乃至１６のいずれか１項に記載の方法。

【外国語明細書】

COMPUTER SYSTEM AND METHOD FOR EXECUTING NETWORK MOBILE CODE WITH REDUCED RUN-TIME MEMORY SPACE REQUIREMENTS

The present invention relates to computer systems and methods for executing programs with reduced run-time memory space requirements. In particular, the present invention pertains to computer systems and methods in a network for executing code transmitted over the network (i.e., network mobile code) so that the run-time memory space requirements of the code is reduced.

BACKGROUND OF THE INVENTION

Computer systems are now being built or configured to take advantage of properties of programs whose code is in an architecture neutral (AN) binary format, hereinafter referred to as AN code. Thus, the AN code of these programs is independent of the specific architecture or platform of the computer system.

The term architecture is defined for the purposes of this document to mean the operating characteristics of a family of computer models. Examples of specific architectures include Macintosh computers, IBM PC compatible computers using the DOS or Windows operating systems, Sun Microsystems computers running the Solaris operating system, and computer systems using the Unix operating system.

The term architecture specific (AS) is defined for the purposes of this document to refer to the requirement that the code of certain programs be in a binary format, hereinafter referred to AS code, for execution only on computer systems with a specific computer architecture. Thus, programs with code written in a conventional programming language (e.g., C language) and compiled for a specific architecture (e.g., IBM compatible PC) can only run on that architecture or emulators of that architecture.

The term architecture neutral (AN) is defined for the purposes of this document to refer to programs whose compiled code can be executed on a variety of computer systems with different architectures. For example, a computer system with a specific architecture can be configured with a Java (a trademark of Sun Microsystems) virtual machine module. The Java virtual machine module enables execution of programs with code written in the Java programming language and compiled into bytecode, hereinafter referred to as Java bytecode, for the instruction set of the Java virtual machine. Java bytecode is independent of the specific architecture of the computer system.

Important features of programs with AN code include their portability. For example, since programs in AN code can be executed on any computer system configured to execute the AN code regardless of the computer system's specific architecture, these programs can be easily transported over a network from one computer system to another. For example, programs compiled into Java bytecode can be executed on any computer system with a Java virtual machine module and can be easily transported over a network from one computer system to another using a HotJava (a trademark of Sun Microsystems) network communications manager.

Furthermore, another important feature related to the portability of programs compiled into Java bytecode is the verifiability of such programs. Specifically, the Java virtual machine module can easily verify that these programs satisfy predefined integrity criteria. Such integrity criteria include stack and data type usage restrictions that ensure that Java bytecode cannot

overflow or underflow the Java virtual machine module's stack and that all instructions in Java bytecode utilize only data whose data type matches the data type restrictions for those instructions. As a result, a program in Java bytecode cannot forge object pointers and generally cannot access system resources other than those which the user has explicitly granted it permission to use.

For these reasons, computer systems are being configured for execution of programs in AN code that are received over a network. In fact, in some cases, such computer systems may not even require a secondary memory (e.g., a hard disk) since the programs are loaded directly into the run-time (i.e., execution-time) memory (e.g., random access memory (RAM)) of the computer system. As a result, the user of such a computer system is freed from the cycle of software purchase, installation, configuration and upgrade that is currently typical of software products.

The just described features of AN code make it particularly attractive for use with small or cheap computer systems that are networked and are loaded with AN code on demand. For example, these kinds of computer systems may be video games, personal digital assistants (PDAs), cellular phones, or other similar computer systems or computer operated devices.

Unfortunately, however, programs in AN code run slower than the same programs in AS code. For example, programs in Java bytecode executed by a Java virtual machine module typically run 2.5 to 20 times as slow as the equivalent program in AS code. Thus, a user of a computer system may find it desirable to receive over a network AS code of a program so that the AS code can be executed on the specific architecture of the computer system. However, to ensure that these programs are securely shipped to a computer system, the network would have to be closed or trusted or these programs have to have embedded digital signatures which can be verified.

Moreover, AS code which has been generated from AN is much larger than the original AN code. For example, AS code generated from Java bytecode is typically 2 to 5 times the size of the Java bytecode. Thus, a fixed amount of run-time memory can hold substantially less compiled AS code than AN code. However, as mentioned previously, AS code is much faster than AN code from which it is generated and may be the only way to achieve adequate performance.

In the just described computer systems, price is extremely important. In practice, one of the most significant costs in building such computer systems is the amount of run-time memory that is required for execution of loaded programs. It is therefore very important to reduce the amount of run-time memory required by these computer systems since such a reduction produces a strong competitive advantage.

Furthermore, in the computer systems of the type described above, it may not be possible to page to secondary memory. In this case, the AN or AS code received over a network could be cached in the run-time memory and flushed when its space in the run-time memory is required. However, when execution of the flushed program is to be continued, the original code must again be downloaded over the network. This significantly affects the execution speed of the program. Furthermore, even in computer systems where it is possible to page to secondary memory, the time required to retrieve the code from the secondary memory may be too costly.

In the present invention, compression and then decompression of the code of a program received over a network is used to reduce the storage cost of the code in the run-time memory. Since this is much faster than flushing code and retrieving it, the execution speed of the code is not significantly affected by compression and decompression.

SUMMARY OF THE INVENTION

In summary, the present invention is a client computer system and associated method in a computer network over which are provided programs with methods. The client computer is capable of executing the programs with reduced run-time memory space requirements. The client computer system comprises a run-time memory, a communications interface, a network communications manager, an execution controller, and a compressor.

The network communications interface receives the methods of the programs and the network communications manager loads uncompressed into available space in the run-time memory the methods when they are received. The execution controller controls execution of the programs so that the methods are invoked and not invoked.

The compressor compresses in the run-time memory compressible ones of the compressed programs that are not invoked. As a result, space is made available in the run-time memory. The compressor also decompresses in available space in the run-time memory decompressible ones of the compressed methods so that they may be invoked.

In one embodiment, the compressor decompresses the decompressible ones of the compressed methods as soon as they are invoked.

In another embodiment, the compressor decompresses the decompressible ones of the compressed methods after a predetermined time interval.

In still another embodiment, the compressor compresses the compressible ones of the uncompressed methods as soon as they are no longer invoked.

In yet another embodiment, the compressor compresses the compressible ones of the uncompressed methods when space in the run-time memory is needed but not available. Moreover, in this embodiment, the client computer system may further comprise a least recently invoked list that lists those of the methods that are currently not invoked in order of least recently invoked method to most recently invoked method. As a result, the compressible ones of the uncompressed methods are the least recently invoked methods In the least recently invoked list when space in the run-time memory is needed but not available.

In yet still another embodiment, the compressor flushes from the run-time memory flushable ones of the compressed methods when space in the run-time memory is needed but not available.

As an alternative to the previous embodiment, the client computer system may further comprise a secondary memory. In this case, the compressor stores in the secondary memory storable ones of the compressed methods when space in the run-time memory is needed but not available. Then the compressor retrieves from the secondary memory retrievable ones of the compressed methods that are to be decompressed.

BRIEF DESCRIPTION OF THE DRAWINGS

Additional goals and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

Figure 1 is a block diagram of a client computer system incorporating the present invention.

Figure 2 is a functional block diagram of the operation of the client computer system.

Figure 3 is a flow chart of the compression method of the client computer system.

Figure 4 is a flow chart of the decompression method of the client computer system.

Figure 5 shows an alternative embodiment of the client computer system.

Figure 6 shows a functional block diagram of the operation of the alternative embodiment of the client computer system.

DETAILED DESCRIPTION OF THE INVENTION

Referring to Figure 1, there is shown a computer network 100 in accordance with the present invention. It includes one or more client computer systems 102, one or more server computer systems 104, and a network communications connection 106.

The client computer systems 102 are connected to the server computer systems 104 via the network communications connection 106. The network communications connection may be a local or wide area network, the Internet, or some other type of network communications connection.

Each server computer system 104 includes a central processing unit (CPU) 110, a user interface 112, a network communications interface 116, and a memory 118. The network communications interface enables each server computer system to communicate with the client computer systems 102 via the network communications connection 106.

The memory 118 of each server computer system 104 stores an operating system 120, a network communications manager (or server) 122, and programs 145. The operating system and communications manager are

all run on the CPU 120. The operating system controls and coordinates running of the network communications manager in response to commands issued by a user with the user interface 112 or received by the network communications interface 116 via the network communications connection 106 from users of the client computer systems 102.

The programs 145 comprise methods 147 and/or 148. For purposes of this document, any discrete fragment or portion of a program that is invoked and not invoked at various times during the execution of the program is considered a method.

The methods 147 of each server computer system 104 contain architecture neutral (AN) code that is independent of the specific architecture (i.e., platform) of the client computer systems 102. These programs are compiled from a specific programming language into the AN code. In the preferred embodiment, these programs are written in the Java programming language and compiled into Java bytecode. Moreover, these programs are included in object classes with methods that form software programs which are programmed in an object-oriented manner.

On the other hand, the methods 148 of each server computer system contain architecture specific (AS) code that is compiled for the specific architecture of the client computer systems 102. As alluded to earlier, these kinds of methods are desirable in that they run faster than the same methods in AN code. As will be explained in greater detail later, it is preferred that the network 100 be a closed and trusted network in which these programs may be securely shipped to the client computers 102 with a high degree of confidence or that these programs have embedded digital signatures which can be verified.

As will also be explained in more detail later, the methods 147 and/or 148 are transmitted upon user request to the client computer systems 102

using the network communications manager 122. Thus, the code of these methods is considered network mobile code.

Each client computer system 102 may be a video game, a personal digital assistant (PDA), a cellular phone, desktop computer, or other computer system or computer operated device which requires a small amount of run-time memory. Furthermore, each client computer system includes a central processing unit (CPU) 126, a user interface 128, a network communications interface 132, a read only memory (ROM) 134, and a run-time random access memory (RAM) 136. The network communications interface enables the client computer system to communicate with the server computer systems 104 via the network communications connection 106.

The RAM 136 of each client computer system 102 stores an operating system 138, a network communications manager 140, a virtual machine module 142, and a code compressor 146 that have all been loaded from the ROM 134. The RAM also stores the programs 145 with methods 147 containing AN code and/or methods 148 containing architecture specific (AS) code that have been downloaded from the server computer systems 104. The operating system, network communications manager, virtual machine module, compiler, compressor, and programs are all executed on the CPU 126. The operating system controls and coordinates execution of the network communications manager, virtual machine module, code compiler, code compressor, and programs in response to commands issued by a user with the user interface 128.

As alluded to earlier, the methods 147 and/or 148 are received from the server computer systems 104 upon user request. These methods are obtained using the network communications manager 122 which is, in the preferred embodiment, a HotJava network communications manager. The network communications manager then loads these methods in the RAM 136.

The code verifier 151 of the virtual machine module 142 verifies that the AN code of the loaded methods 147 meets predefined integrity criteria. As mentioned earlier, this may include stack and data type usage restrictions to ensure that loaded methods cannot overflow or underflow the virtual machine module's stack and that all instructions utilize only data whose data type matches the data type restrictions for those instructions. In the preferred embodiment, the virtual machine module is a Java virtual machine module.

However, in the case of the received methods 148 with AS code, the code verifier 151 cannot be used to directly verify their integrity. Thus, to verify their integrity indirectly, the network 100 may be a closed and trusted network in which these methods may be securely shipped to the client computers 102 with a high degree of confidence. Alternatively, if the network 100 is not secure, these programs may have embedded digital signatures which enable the network communications manager 140 to verify that they are from a trusted source.

The execution controller 153 of the virtual machine module 142 controls execution of the programs 145. In particular, the execution controller interprets the AN code of the methods 147 for execution on the specific architecture of the client computer system 102 and enables these methods to call the methods 148 containing AS code for execution on the specific architecture. The execution 149 data generated during the execution of the programs is stored in the RAM 136. In addition, if the network communications manager 140, code compiler 144, and/or code compressor 146 are in AN code, then the execution controller controls execution of them as well.

Furthermore, in order to keep the RAM space requirements of the client computer system 102 low, the code compressor 146 compresses and decompresses in the RAM 136 the code of the methods 147 and/or 148 at various times. This is done for the methods that are compressible and decompressible because they respectively satisfy predefined compression

and decompression criteria 152 and 154 that are stored in the RAM 136 and inputted by the user with the user interface 128. The compression and decompression criteria are more fully described later and are, in some embodiments of the present invention, user selectable and/or tunable (using the user interface 128) from a set of predefined memory management strategies.

The storage and invocation status of the methods 147 and/or 148 is maintained with the method status data structures 155. The method status data structures are updated by the network communications manager 140, the execution controller 153, and the code compressor 146.

Figure 2 shows a functional block diagram of the operation of each of the client computer systems 102 in compressing and decompressing in the RAM 136 the methods 147 and/or 148. In addition, Figures 3 and 4 respectively show the preferred compression and decompression methods 300 and 400.

Referring to Figures 1-3, when a user requests execution of one of the programs 145 of one of the server computer systems 104, the user's client computer system 102 obtains the requested program from the server computer system (step 302 of Figure 3). This is done when the user issues a command with the user interface 128 to download and execute the program from the server computer system. In response, the operating system 120 calls the network communications manager 140 which generates a message indicating that such a request has been made. The network communications interface 132 then transmits the message to the server computer system.

The network communications interface 116 of the server computer system 104 receives the transmitted message. In response, the network communications manager 122 of the server computer system provides the methods 147 and/or 148 of the requested program 145 to the network

communications interface which then transmits the methods to the user's client computer system 102.

The methods 147 and/or 148 that were transmitted are received by the network communications interface 132 of the user's client computer system 102. In response, the network communications manager 140 then determines if enough space in the RAM 136 is available for loading the received methods (decision step 304 of Figure 3). If there is, then the network communications manager loads the code of these methods uncompressed into the available space in the RAM (step 306 of Figure 3). As a result, these methods are loaded into the RAM along with previously loaded programs methods 147 and/or 148 of other programs 145. In loading these methods, the network communications manager updates the method storage status table 200 of the method status data structures 155 to identify the methods and the corresponding pointers to the memory spaces in the RAM 136 occupied by these methods. Furthermore, the network communications manager updates the program status table to indicate that the code of the methods is uncompressed (U).

The network communications manager 140 than invokes the code verifier 151 of the virtual machine module 142. In response, the code verifier verifies that the AN code of any methods 147 that were just loaded meets the predefined integrity criteria discussed earlier (step 307 of Figure 3).

The program 145 whose methods 147 and/or 148 were just loaded is then executed under the control of the execution controller (step 314 of Figure 3) along with any previously loaded programs. As the programs are executed, their methods are invoked and not invoked at various times during their execution. As mentioned previously, the execution controller interprets the AN code of the methods 147 for execution on the specific architecture of the user's client computer system 102 and enables these methods to call the methods 148 containing AS code for execution on the specific architecture.

Each of the loaded methods 147 and/or 148 may not be invoked at various times because it may have to wait for data, may have been put to sleep, etc... When the execution controller determines that this has occurred, it adds the method to the least recently invoked (LRI) list 202 of the program status data structures 155 to indicate that it is currently not invoked. The methods in the LRI list are listed from most recently invoked to least recently invoked.

As indicated earlier, in order to reduce the space requirements of the RAM 136, the code of each loaded method 147 and/or 148 for which the predefined compression criteria 152 is satisfied is compressed by the code compressor 146. In the preferred embodiment, the compression criteria specifies that the code of a compressible method 147 or 148 be compressed at the time when (1) space in the RAM 136 is needed but not available, and (2) the method is the least recently invoked method in the LRI list 202 that has not yet had its code compressed.

When the network communications manager 140 determines that space in the RAM 136 is not available to load one or more of the methods 147 and/or 148 received by the server computer systems 104 (decision step 304 of Figure 3), then it invokes the code compressor 146. In response, the code compressor compresses the code of the least recently invoked method(s) in the LRI list 202 whose code has not yet been compressed until enough space has been made available (step 316 of Figure 3). The code compressor also updates the method storage status table 200 to identify the corresponding pointer(s) to the memory space(s) of the compressed code of the method(s) and to indicate that the code of the method(s) has been compressed (C). The network communications manager then loads the method(s) for which space has been made available into the available space, as described earlier (step 306 of Figure 3).

The code compressor 146 may use any fast data compression technique well known to those skilled in the art. Moreover, the code

compressor may use separate compression techniques for optimal compression of the AN code of the methods 147 and the AS cod of the methods 148.

Furthermore, while the loaded methods 147 and/or 148 are invoked, they generate execution data. For any of these methods which the execution controller 153 determines is invoked (decision step 320 of Figure 3) and for which space in the RAM 136 is available to store execution data (decision step 322), the execution controller stores the execution data in the RAM (step 324 of Figure 3).

However, for each of the loaded methods 147 and/or 148 which the execution controller 153 determines is invoked (decision step 320 of Figure 3) and for which space in the RAM is not available but needed to store execution data (decision step 322), the execution controller invokes the code compressor 146. As in the earlier situations where space in the RAM is needed, the code compressor compresses the code of the least recently invoked method(s) in the LRI list 202 whose code has not yet been compressed until enough space has been made available (step 326 of Figure 3). And, as described earlier, the code compressor updates the method storage status table 200 to identify the corresponding pointer(s) to the memory space(s) of the compressed code of the method(s) and to indicate that the code of the method(s) has been compressed (C). The execution controller then stores the execution data in the space made available in the RAM (step 324 of Figure 3), as described earlier, and execution of the program with the method(s) whose execution data was stored continues (step 314 of Figure 3).

Moreover, referring to Figures 1, 2, and 4, when space in the RAM 136 is available, each loaded method 147 and/or 148 that is compressed and for which the predefined decompression criteria 154 is satisfied is decompressed by the code compressor 146. In the preferred embodiment, the predefined decompression criteria 154 for decompressing a

decompressible method's code simply specifies that the code of the method is compressed and is to be decompressed at the time when the method is to be invoked again.

Thus, whenever the execution controller 153 determines that one of the loaded methods 147 and/or 148 is to be invoked (decision step 402 of Figure 4), it determines whether the code of this method is compressed (decision step 404 of Figure 4). This is determined from the method storage status table 200. If the method's code isn't compressed, then the method is invoked by the execution controller (step 406 of Figure 4). When it is no longer invoked, its code may be compressed by the code compressor 146 in the manner described earlier.

However, if the code of the method 147 or 148 that is to be invoked is compressed, then the execution controller invokes the code compressor 146 to decompress the method's code. The code compressor determines if there is enough space available in the RAM 136 to decompress the code (decision step 408 of Figure 4). If there is enough space available, the code compressor decompresses the code in the available space (step 410 of Figure 4). In doing so, it updates the method storage status table 200 to identify the corresponding pointer(s) to the memory space(s) of the uncompressed code and to indicate that the method's code is now uncompressed (U).

However, if there is not enough space available, then the code compressor 146 compresses the code of the least recently executed method(s) in the LRI list 202 whose code has not yet been compressed until space has been made available (step 412 of Figure 4). This is done in the manner described earlier. After the space has been made available, the code of the method 147 or 148 that is to be decompressed is decompressed by the code compressor in the available space and then invoked by the execution controller 153 in the manner just described (steps 410 and 406 of Figure 4).

In view of the foregoing, it is clear that the present invention provides a reduction in run-time memory space while saving execution speed. This is due to the fact that by compressing the loaded methods 147 and/or 148, they do not need not be flushed from the RAM 136 and re-downloaded from the server computer systems 104 when space in the RAM is needed. However, as those skilled in the art will recognize, other alternative embodiments could be implemented to provide similar benefits.

Specifically, the compression criteria 152 described earlier specified that the code of least recently executed methods 147 and/or 148 with uncompressed code would be compressed when space in the RAM 136 was needed but not available. However, the compression criteria may be selected by the user from a broad range of options and based on a number of conditions specific to the user's client computer system 102. For example, the compression criteria may simply specify that the code of each method is to be compressed as soon as the method is no longer invoked. Or, it may specify that the code of certain methods is to be compressed lazily whenever time is available for doing so. As an additional variation of any of the foregoing, the compression criteria may specify that only the code of methods of a specific size or type is to be compressed.

Moreover, as was stated earlier, the decompression criteria 154 specified that a method 147 or 148 whose code is compressed code would have its code decompressed as soon as the method was to be invoked. However, the decompression criteria could specify that the compressed code be decompressed after a predetermined time interval has expired. In this case, the data compressor would include a timer to time the time interval. In one example, this technique could be used for a method that is being put to sleep for a known time interval so that the method will be compressed for this time interval and then decompressed just prior to when it is to be awakened. Or, in another example, the technique could be used for a method that is waiting for data where the time interval over which the method is compressed

would be selected so as to predict when the data will become available for the method.

In another embodiment, the compression criteria 152 may also include flushing criteria specifying when the loaded methods 147 and/or 148 are to be flushed from the RAM 136. For example, the flushing criteria may indicate that if there are no more loaded methods in the RAM that are not invoked and uncompressed, then the least recently executed method(s) in the LRI list 202 with compressed code are flushable and are to be flushed from the RAM 136 until enough space is made available. As a result, when a method is flushed from the RAM and then is to be executed again, the network communications manager 140 must re-download the program from the server computer system 104 that supplied the program in the manner discussed earlier. The execution controller 153 updates the program storage status table 200 by removing reference to any flushed methods.

Since flushing loaded methods 147 and/or 148 from the RAM 136 is costly in terms of execution speed, a secondary memory 500 could be used to store the methods that would otherwise be flushed, as shown in Figures 5 and 6. In this case, the compression criteria 152 discussed earlier would include secondary storage criteria that would be similar to the flushing criteria just discussed and used to store methods that are storable because they satisfy the secondary storage criteria. However, in this case, the code compressor 146 would update the pointers in the program storage status table 200 to point to these methods in the secondary memory. Furthermore, for the methods that are retrievable in that their code is compressed, stored in the secondary memory, and is to be decompressed; the code of the methods would be retrieved from the secondary memory and then decompressed in the RAM 136 in the manner described earlier.

Moreover, in an embodiment where the client computer system 102 includes a secondary memory 500 (e.g. a networked desktop computer), the methods 147 and/or 148 could be downloaded from the server computer

systems 104 to the secondary memory. Then, these methods could be loaded directly into the RAM 136 from the secondary memory rather then from the server computer systems 104. Additionally, in such an embodiment, the operating system 138, the network communications manager 140, the virtual machine module 142, and the code compressor 146 could be stored in the secondary memory and loaded from there into the RAM.

In still another embodiment, the operating system 138, the network communications manager 140, the virtual machine module 142, and the code compressor 146 could be downloaded from one of the server computer systems 104 into the RAM 136 of the client computer system 102. This would be done in a similar manner as that described earlier for the methods 147 and/or 148 of a server computer system.

In yet another embodiment, the virtual machine module 142 is actually implemented in a silicon chip and serves as the CPU 126 of the client computer system 102. In this case, the methods 147 with AN code are not interpreted for a specific architecture and are instead executed directly. In this embodiment, methods 148 with AS code would not be utilized.

Finally, while the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. In a computer network over which is provided programs with methods, a client computer system for executing the programs with reduced run-time memory space requirements, the client computer system comprising:

an run-time memory;

a network communications interface that receives the methods;

a network communications manager that loads the methods uncompressed into available space in the run-time memory when received;

an execution controller that controls execution of the programs, whereby the methods are invoked and not invoked at different times;

a compressor that (A) compresses in the run-time memory compressible ones of the uncompressed methods that are not invoked, whereby space is made available in the run-time memory, and (B) decompresses in available space in the run-time memory decompressible ones of the compressed methods so that the decompressible ones of the compressed methods may be invoked.

2. The client computer system of claim 1 wherein the compressor decompresses the decompressible ones of the compressed methods as soon as the decompressible ones of the compressed methods are to be invoked.

3. The computer system of claim 1 wherein the compressor decompresses the decompressible ones of the compressed methods after a predetermined time interval.

4. The client computer system of claim 1, 2 or 3, wherein the compressor compresses the compressible ones of the uncompressed methods as soon as the compressible ones of the uncompressed methods are no longer invoked.

5. The client computer system of claim 1, 2 or 3, wherein the compressor compresses the compressible ones of the uncompressed methods when space in the run-time memory is needed but not available.

6. The client computer system of any of claims 1 through 5 further comprising:

a least recently invoked list that lists those of the methods that are currently not invoked in order of least recently invoked method to most recently invoked method;

the compressible ones of the uncompressed methods are the least recently executed methods in the least recently executed list that are not compressed when space in the run-time memory is needed but not available.

7. The client computer system of any of claims 1 through 6, wherein:

the methods include methods in architecture neutral code that is independent of the architecture of the client computer system; and

the client computer system further comprises a virtual machine module that includes the execution controller and enables execution of the methods in the architecture neutral code.

8. The client computer system of any of claims 1 through 7, wherein the compressor flushes from the run-time memory flushable ones of the compressed methods when space in the run-time memory is needed but not available.

9. The client computer system of any of claims 1 through 8 further comprising:

a secondary memory;

the compressor (A) stores in the secondary memory storable ones of the compressed methods when space in the run-time memory is needed but not available, and (B) retrieves from the secondary memory retrievable ones of the compressed methods that are to be decompressed.

10. In a computer network over which is provided programs with methods, a method of executing the programs with reduced run-time memory space requirements, the method comprising the steps of:

providing an run-time memory;

receiving the methods;

loading uncompressed into available space in the run-time memory the methods when they are received;

executing the programs, whereby the methods are invoked and not invoked at different times;

compressing in the memory compressible ones of the uncompressed methods that are not invoked, whereby space is made available in the run-time memory; and

decompressing into available space in the run-time memory decompressible ones of the compressed methods so that the decompressible ones of the compressed methods may be invoked.

11. The method of claim 10 wherein the decompressing step includes decompressing the decompressible ones of the compressed methods as soon as the decompressible ones of the compressed methods are to be invoked.

12. The method of claim 10 wherein the decompressing step includes decompressing the decompressible ones of the compressed methods after a predetermined time interval.

13. The method of claim 19, 11 or 12, wherein the compressing step includes compressing the compressible ones of the uncompressed methods as soon as the compressible ones of the compressed methods are no longer invoked.

14. The method of claim 10, 11 or 12, wherein the compressing step includes compressing the compressible ones of the uncompressed methods when space in the run-time memory is needed but not available.

15. The method of any of claims 10 through 14 further comprising the steps of:

providing a least recently invoked list that lists those of the methods that are currently not invoked in order of least recently invoked method to most recently invoked method;

the compressible ones of the uncompressed methods are the least recently invoked methods in the least recently invoked list that are not compressed when space in the run-time memory is needed but not available.

16. The method of any of claims 10 through 15 further comprising the step of flushing from the run-time memory flushable ones of the compressed methods when space in the run-time memory is needed but not available.

17. The method of any of claims 10 through 16 further comprising the steps of:

providing a secondary memory;

storing in the secondary memory storable ones of the compressed methods when space in the run-time memory is needed but not available; and

retrieving from the secondary memory retrievable ones of the compressed methods that are to be decompressed.

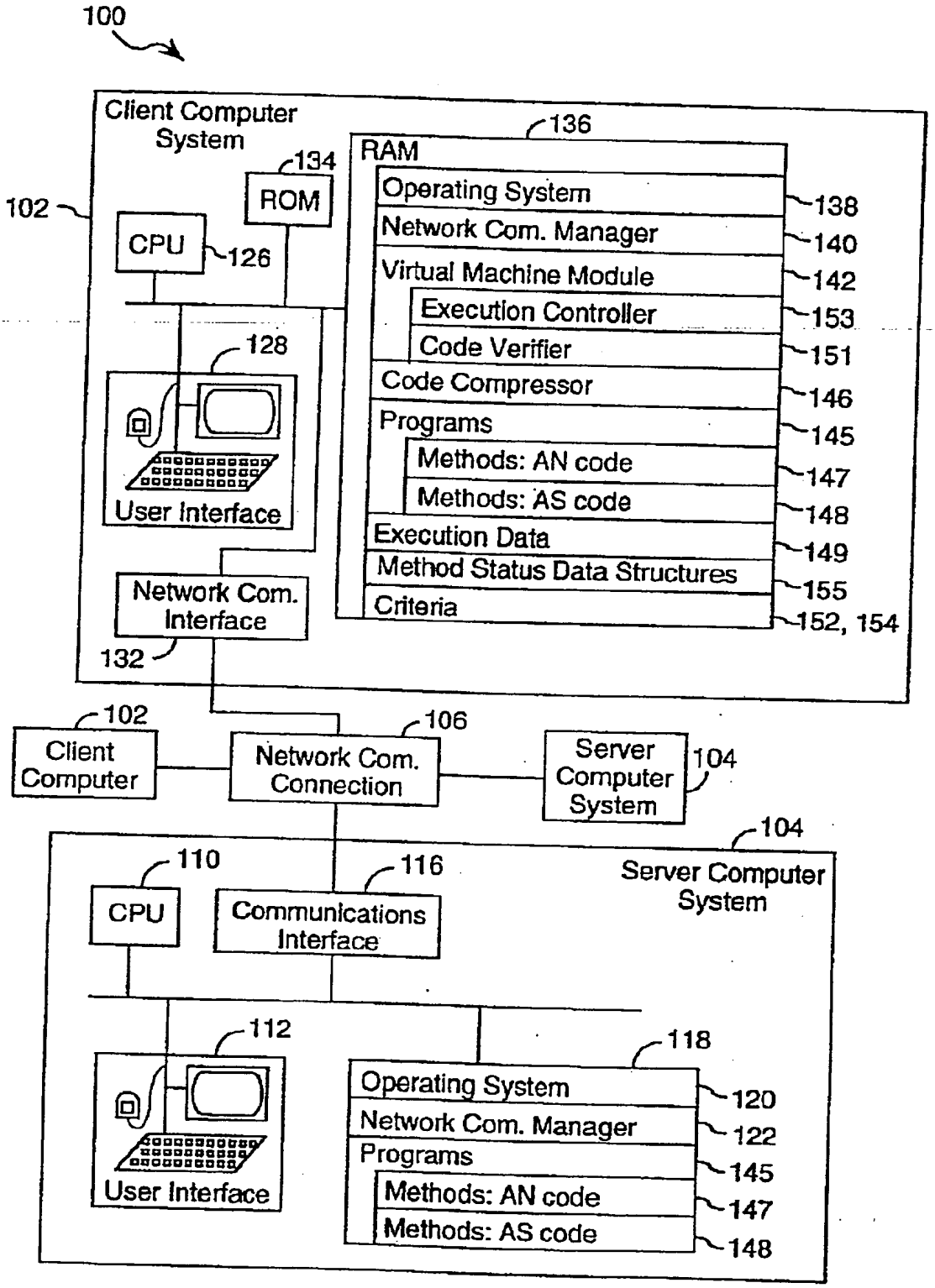

100
Client Computer System
102
134
ROM
CPU
126
128
User Interface
Network Com. Interface
132
136
RAM
Operating System
138
Network Com. Manager
140
Virtual Machine Module
142
Execution Controller
153
Code Verifier
151
Code Compressor
146
Programs
145
Methods: AN code
147
Methods: AS code
148
Execution Data
149
Method Status Data Structures
155
Criteria
152, 154
102
Client Computer
106
Network Com. Connection
Server Computer System
104
104
Server Computer System
110
CPU
116
Communications Interface
112
User Interface
118
Operating System
120
Network Com. Manager
122
Programs
145
Methods: AN code
147
Methods: AS code
148

Figure 1

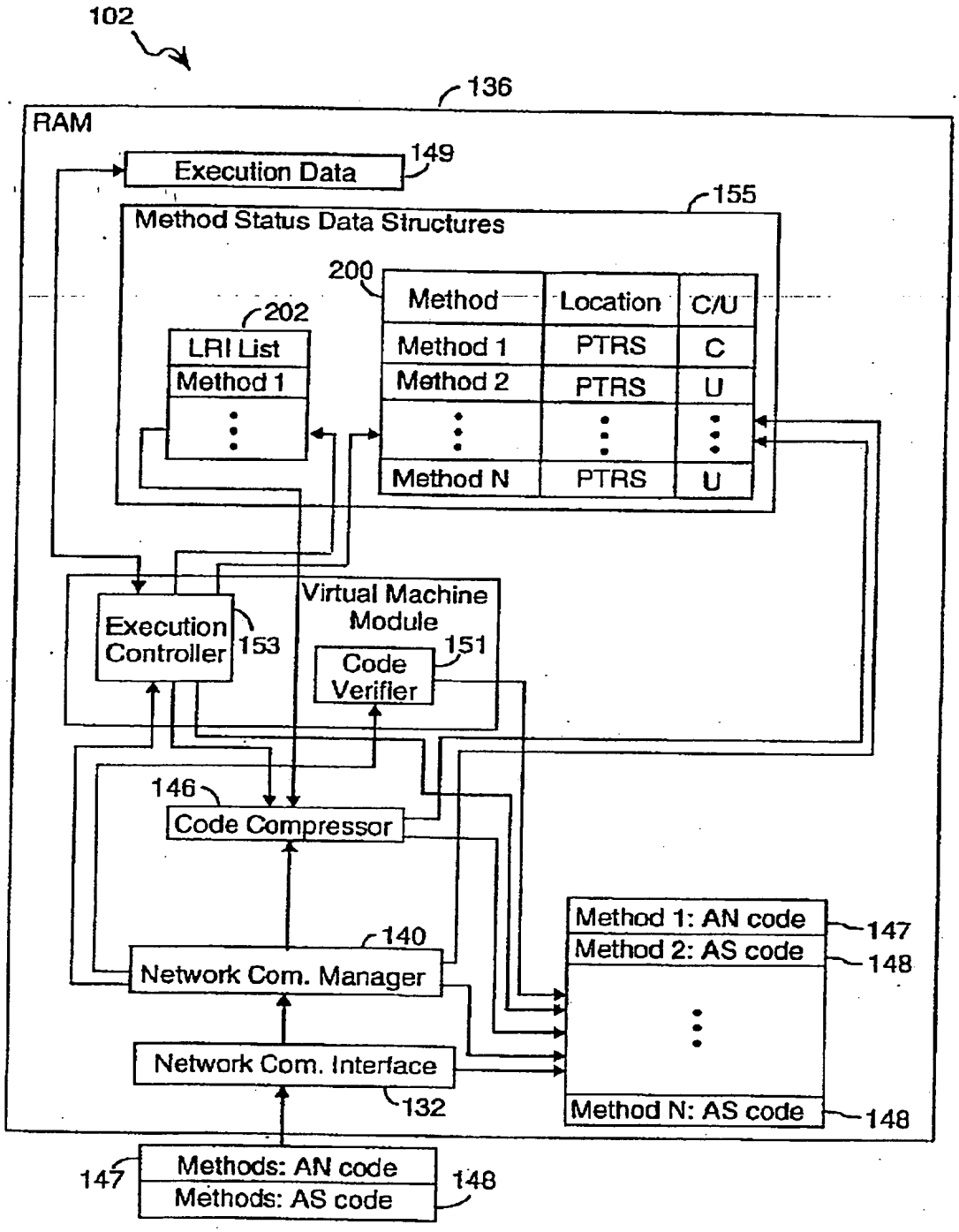

102
136
RAM
Execution Data
149
155
Method Status Data Structures
200
Method
Location
C/U
202
LRI List
Method 1
Method 1
PTRS
C
Method 2
PTRS
U
Method N
PTRS
U
Virtual Machine Module
Execution Controller
153
Code Verifier
151
146
Code Compressor
140
Network Com. Manager
Network Com. Interface
132
Method 1: AN code
147
Method 2: AS code
148
Method N: AS code
148
147
Methods: AN code
Methods: AS code
148

FIGURE 2

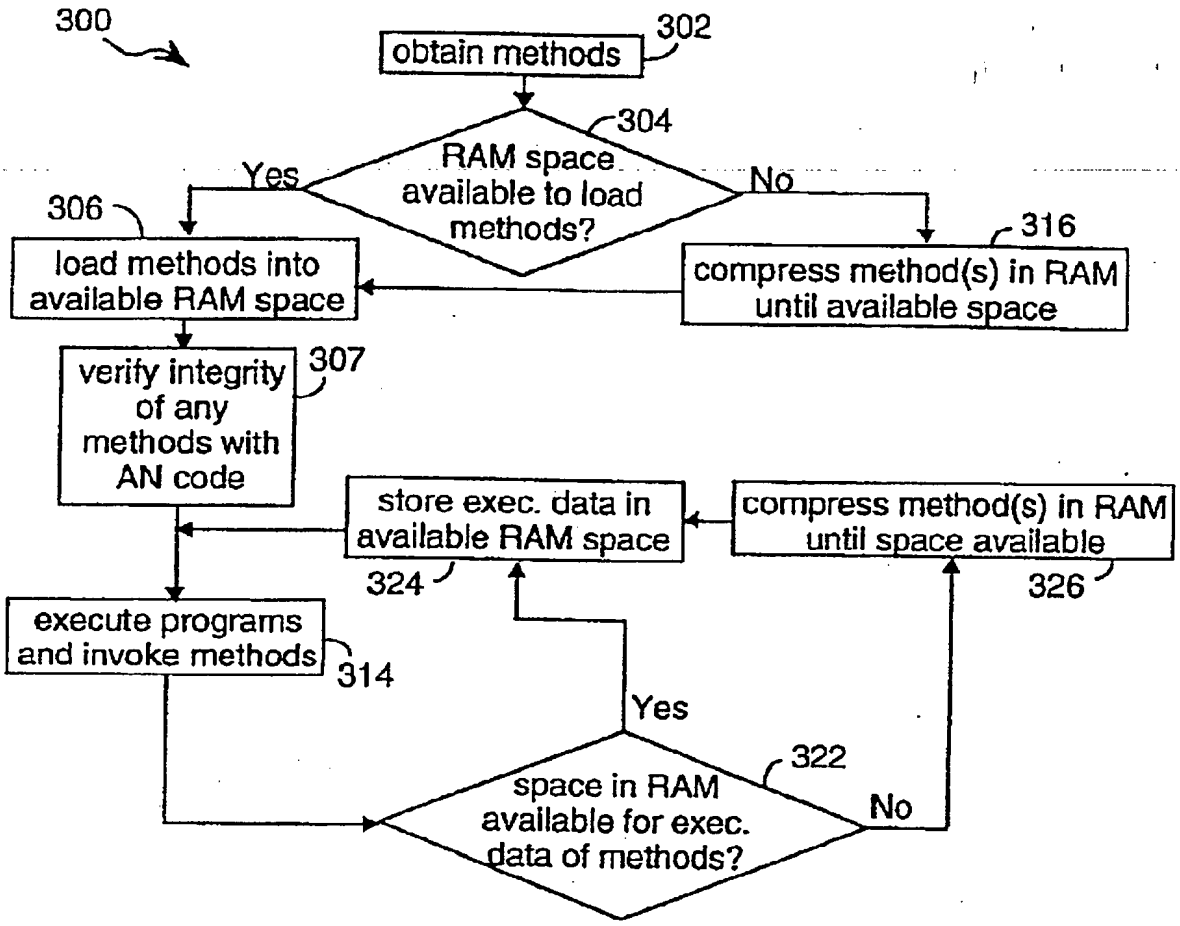

300
obtain methods
302
RAM space available to load methods?
304
Yes
No
306
load methods into available RAM space
316
compress method(s) in RAM until available space
verify integrity of any methods with AN code
307
store exec. data in available RAM space
compress method(s) in RAM until space available
324
326
execute programs and invoke methods
314
Yes
322
space in RAM available for exec. data of methods?
No

FIGURE 3

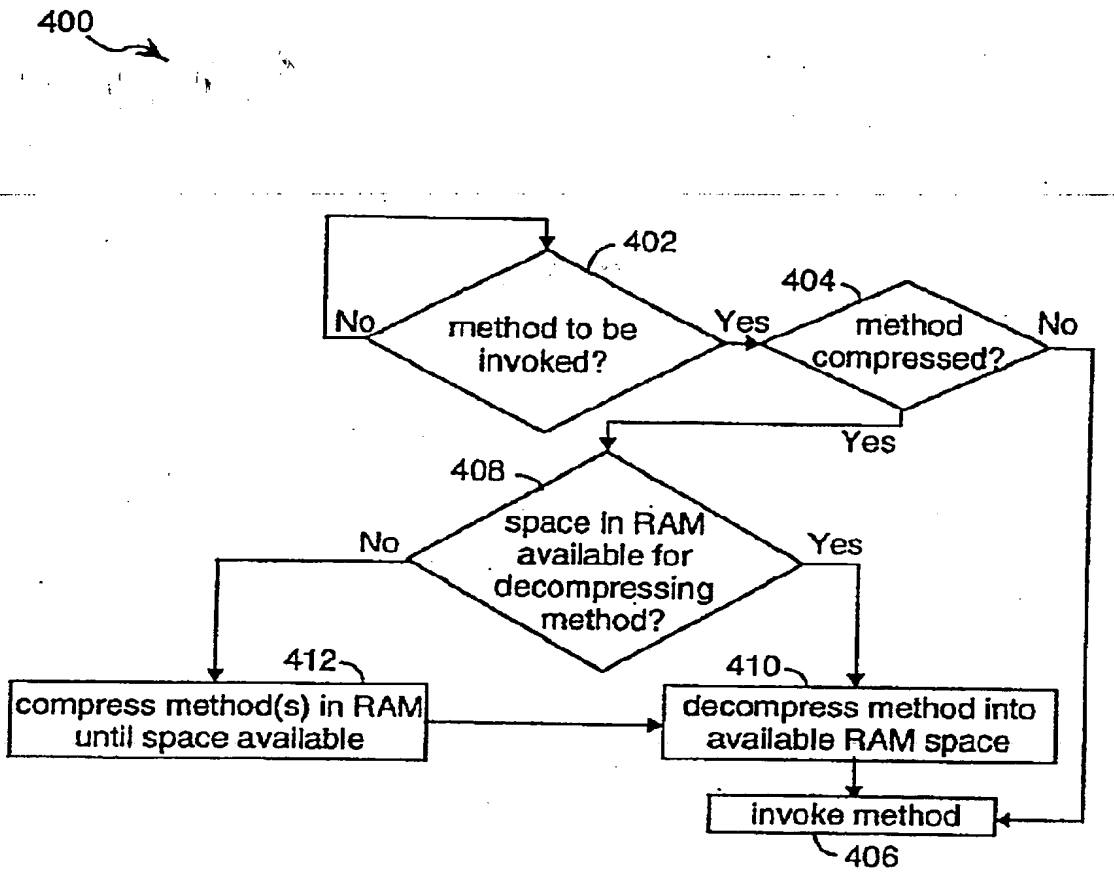

400
402
method to be invoked?
No
Yes
404
method compressed?
No
Yes
408
space in RAM available for decompressing method?
No
Yes
412
compress method(s) in RAM until space available
410
decompress method into available RAM space
invoke method
406

FIGURE 4

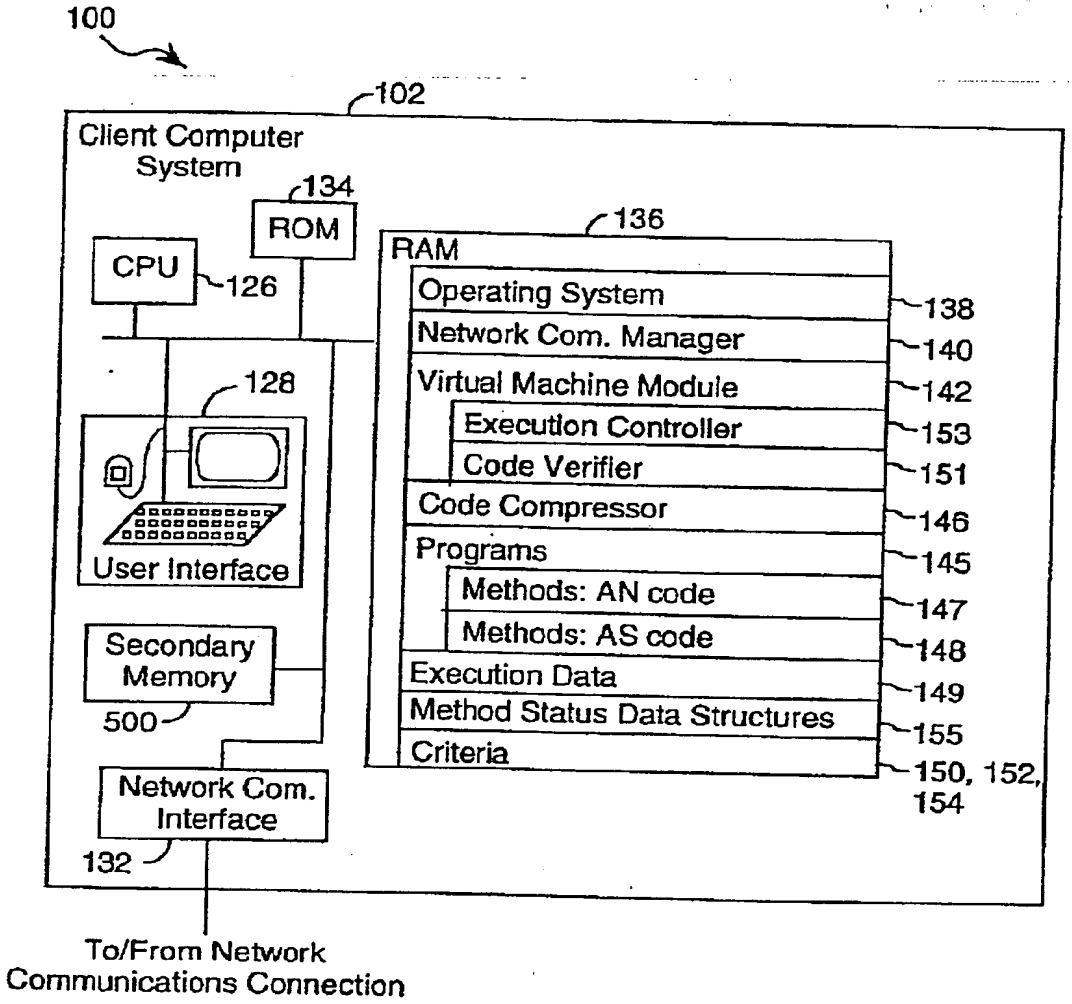

100
102
Client Computer System
134
ROM
136
RAM
CPU
126
Operating System
138
Network Com. Manager
140
Virtual Machine Module
142
Execution Controller
153
Code Verifier
151
128
Code Compressor
146
Programs
145
User Interface
Methods: AN code
147
Methods: AS code
148
Secondary Memory
500
Execution Data
149
Method Status Data Structures
155
Criteria
150, 152, 154
Network Com. Interface
132
To/From Network Communications Connection

FIGURE 5

6

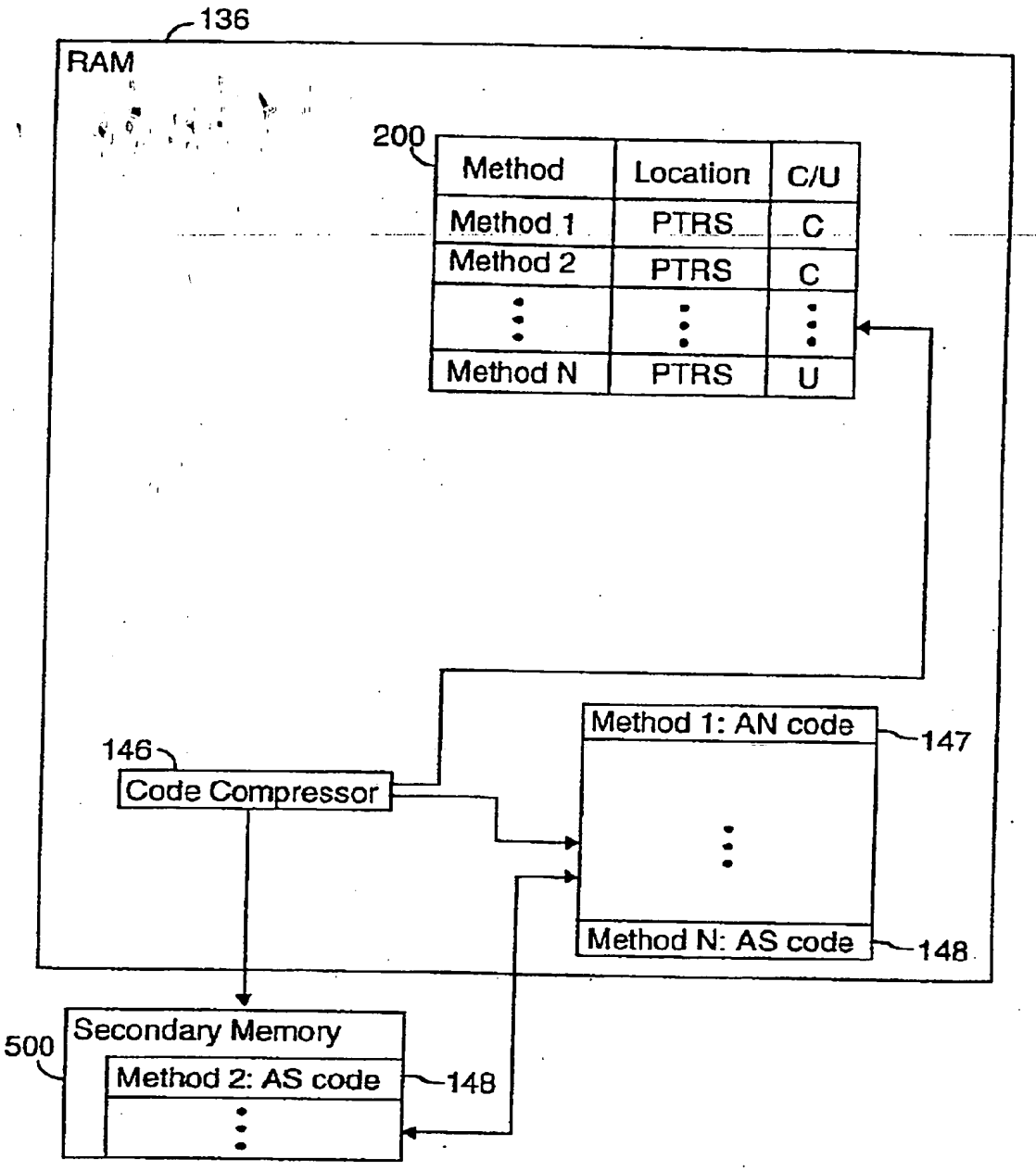

136
RAM
200
Method
Location
C/U
Method 1
PTRS
C
Method 2
PTRS
C
Method N
PTRS
U
146
Code Compressor
Method 1: AN code
147
Method N: AS code
148
500
Secondary Memory
Method 2: AS code
148

FIGURE 6

ABSTRACT

A client computer system and associated method in a computer network over which are provided programs with methods. The client computer is capable of executing the programs with reduced run-time memory space requirements. Specifically, a network communications interface receives the methods of the programs and a network communications manager loads uncompressed in available space in the run-time memory the methods when they are received. An execution controller controls execution of the programs so that the methods are invoked and not invoked at various times during execution of the programs. A compressor compresses in the memory compressible ones of the uncompressed methods that are not invoked so that space is made available in the run-time memory. The compressor also decompresses in available space in the run-time memory decompressible ones of the methods so that they may be invoked.